ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Μονότονος καθαρισμός ενός μολυσμένου τυχαίου δυαδικού δέντρου με τη χρήση σχεδόν ελάχιστου αριθμού ερευνητών

Τελώνης Γεώργιος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
Επιβλέπων
Μάρκου Ευριπίδης
Επίκουρος Καθηγητής

Λαμία, 2015

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗ ΒΙΟΙΑΤΡΙΚΗ

Μονότονος καθαρισμός ενός μολυσμένου τυχαίου δυαδικού δέντρου με τη χρήση σχεδόν ελάχιστου αριθμού ερευνητών

Τελώνης Γεώργιος

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
Επιβλέπων
Μάρκου Ευριπίδης
Επίκουρος Καθηγητής

Λαμία, 2015

Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [1], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

1. *Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί* ***χωρίς να τα περικλείω σε εισαγωγικά*** *και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

2. *Δέχομαι ότι η αυτολεξεί παράθεση χωρίς εισαγωγικά, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. *Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια.*

4. *Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.*

Ημερομηνία: 2 Νοεμβρίου 2015

Ο Δηλών

(Υπογραφή)

[1] Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.

Μονότονος καθαρισμός ενός μολυσμένου τυχαίου δυαδικού δέντρου με τη χρήση σχεδόν ελάχιστου αριθμού ερευνητών

Τελώνης Γεώργιος

Τριμελής Επιτροπή:

Μάρκου Ευριπίδης, Επίκουρος Καθηγητής

Αδάμ Μαρία, Επίκουρος Καθηγητής

Πλαγιανάκος Βασίλειος, Αναπληρωτής Καθηγητής

Ευχαριστίες

Πάνω από όλα θα ήθελα να ευχαριστήσω θερμά τον επίκουρο καθηγητή του τμήματος κ. Μάρκου Ευριπίδη, για την ανάθεση της συγκεκριμένης πτυχιακής εργασίας, για τη στήριξη και τη διαρκή καθοδήγηση που μου παρείχε, καθώς επίσης και για τις γνώσεις που μοιράστηκε μαζί μου.
Θα ήθελα επίσης να ευχαριστήσω και τα μέλη της επιτροπής, την επίκουρο καθηγητή κ. Αδάμ Μαρία και τον αναπληρωτή καθηγητή κ. Πλαγιανάκο Βασίλειο για τα εποικοδομητικά τους σχόλια.

Περίληψη

Ο υπολογισμός του ελάχιστου αριθμού πρακτόρων (ερευνητών) που απαιτούνται για να καθαρίσουν ένα μολυσμένο δίκτυο είναι ένα διάσημο πρόβλημα στη Θεωρητική Πληροφορική. Η επίλυση αυτού του προβλήματος, που συνήθως ονομάζεται Graph Searching, σε ένα δίκτυο έχει σημαντικές επιπτώσεις στον υπολογισμό ενός σημαντικού χαρακτηριστικού του δικτύου, το οποίο ονομάζεται Pathwidth.
Πιο αναλυτικά, στο πρόβλημα Graph Searching δίνεται ένα γράφημα το οποίο είναι μολυσμένο και ζητείται ο ελάχιστος αριθμός πρακτόρων που μπορούν να καθαρίσουν το γράφημα. Ανάλογα με τις επιτρεπόμενες στρατηγικές κίνησης των πρακτόρων και τους επιτρεπόμενους τρόπους καθαρισμού των ακμών του δικτύου, έχουν εμφανιστεί αρκετές παραλλαγές του προβλήματος. Η επίλυση του αντίστοιχου προβλήματος απόφασης σε οποιαδήποτε παραλλαγή για γενική τοπολογία δικτύου, έχει αποδειχθεί ότι είναι υπολογιστικά δύσκολη (NP-hard).
Το πρόβλημα απόφασης που αντιστοιχεί στον υπολογισμό του εύρους μονοπατιών (Pathwidth) ενός γενικού δικτύου έχει επίσης αποδειχθεί ότι είναι υπολογιστικά δύσκολο (NP-hard). Επιπλέον, δεν έχει βρεθεί γρήγορος αλγόριθμος (πολυωνυμικού χρόνου ως προς το μέγεθος του γραφήματος) που να υπολογίζει μια καλή προσέγγιση για το Pathwidth του γραφήματος. Είναι γνωστό όμως πως η βέλτιστη λύση του προβλήματος Graph Searching σε ένα γράφημα θα μας έδινε αμέσως μια καλή προσέγγιση για το Pathwidth του γραφήματος. Δυστυχώς, στις τοπολογίες γραφημάτων που έχουν μελετηθεί μέχρι τώρα, για τις οποίες ο υπολογισμός του Pathwidth είναι υπολογιστικά δύσκολος, όλες οι παραλλαγές του Graph Searching εκτός από μία είναι επίσης υπολογιστικά δύσκολες. Αυτή η σχετικά νέα παραλλαγή του Graph Searching, ονομάζεται Exclusive Graph Searching και είναι το αντικείμενο μελέτης αυτής της εργασίας.
Ειδικότερα, ασχολούμαστε με την επίλυση του Exclusive Graph Searching σε τοπολογίες δυαδικών δέντρων χρησιμοποιώντας μονότονες στρατηγικές. Ενώ είναι γνωστός ένας αλγόριθμος πολυωνυμικού χρόνου που λύνει το πρόβλημα σε γενικά δέντρα, η υπολογιστική του πολυπλοκότητα είναι αρκετά μεγάλη (αν και πολυωνυμική) και ο ίδιος ο αλγόριθμος είναι αρκετά πολύπλοκος. Εδώ δίνουμε έναν αρκετά πιο απλό, βέλτιστο αλγόριθμο για δυαδικά δέντρα που υπολογίζει τον ελάχιστο αριθμό ερευνητών και καθαρίζει σε γραμμικό χρόνο το δέντρο. Αποδεικνύουμε αναλυτικά την ορθότητα και την πολυπλοκότητα του αλγόριθμου καθώς και ότι ο αριθμός των ερευνητών που χρησιμοποιεί είναι ο ελάχιστος.
Η μελέτη και επίλυση του προβλήματος Graph Searching, εκτός από την εφαρμογή στον υπολογισμό του Pathwidth, έχει επίσης σημαντικές εφαρμογές στα μοντέλα προσομείωσης για την κατάσβεση μιας πυρκαγιάς και γενικά στα μοντέλα προσομείωσης για την εξάλειψη

μόλυνσης σε ένα δίκτυο.

Abstract

The computation of the minimum number of agents (searchers) needed to clean a contaminated network is a popular problem in Theoretical Computer Science. The solution of this problem, known as Graph Searching, in a network would help the computation of Pathwidth, which is an important property of the network.

The instance of the Graph Searching problem consists of a contaminated graph and the goal is to compute the minimum number of agents needed to clean the graph. Depending on the movement strategy of the agents and how the edges of the network can be cleaned, a few variations of the problem have appeared in the literature. The solution of the corresponding decision problem in any such variation for arbitrary networks has been proved to be computationally hard (NP-hard).

The corresponding decision problem of the computation of Pathwidth of an arbitrary network has been also proved to be computationally hard (NP-hard). Moreover, no efficient algorithm (i.e., polynomial-time with respect to the size of the graph) for computing a good approximation of Pathwidth has been found. However, it is known that an optimal solution for Graph Searching in a graph would immediately lead to a good approximation for the Pathwidth of this graph. Unfortunately, in all graph topologies studied so far, for which the computation of Pathwidth is NP-hard, all variations of Graph Searching except one, are also NP-hard. We study here this, relatively new, variation of Graph Searching which is called Exclusive Graph Searching.

More specifically, we study the solvability of Exclusive Graph Searching in binary tree topologies, using monotone strategies. While a polynomial-time optimal algorithm for the problem in arbitrary trees is known, its computational complexity is high (although polynomial) and the algorithm itself is rather technical and complex. We present here a more simple and intuitive algorithm for binary trees which decontaminates in linear time the network using a minimum number of agents. We rigorously prove the correctness and computational complexity of the algorithm. We also prove that the number of agents used by the algorithm is minimum.

The study and solution of the Graph Searching problem, apart from its application to Pathwidth computation, has also important applications to simulation models for distinguishing a fire and contamination prevention in networks.

Περιεχόμενα

Περίληψη **6**

Abstract **8**

1 Εισαγωγή **13**
1.1 Τυπικός ορισμός του παιχνιδιού 14
1.2 Παραλλαγές 14
1.3 Ιδιότητες του Graph Searching 15
1.4 Σχετική δουλειά 16

2 Επισκόπηση αλγορίθμου **19**
2.1 Η διαδικασία της επισήμανσης 19
2.2 Η διαδικασία της τοποθέτησης 20
2.3 Η διαδικασία του καθαρισμού, φάση 1 20
2.4 Η διαδικασία του καθαρισμού, φάση 2 21

3 Επισήμανση κόμβων **23**
3.1 Ο αλγόριθμος postorderLabeling 23
3.2 Παράδειγμα Εκτέλεσης αλγορίθμου 23
3.3 Ανάλυση Αλγορίθμου 24
3.3.1 Ανάλυση ορθότητας 24
3.3.2 Απόδειξη πολυπλοκότητας 25

4 Εισαγωγή ερευνητών στο δυαδικό δέντρο **27**
4.1 Ο αλγόριθμος putSearchers 27
4.2 Παράδειγμα εκτέλεσης αλγορίθμου 28
4.3 Ανάλυση αλγορίθμου 32
4.3.1 Απόδειξη ορθότητας 32
4.3.2 Απόδειξη πολυπλοκότητας 34
4.4 Αριθμός ερευνητών, κάτω φράγμα 34

5 Καθαρισμός, φάση 1 **37**
5.1 Ο αλγόριθμος postorderCleaning 37
5.2 Παράδειγμα εκτέλεσης αλγορίθμου 37

5.3 Ανάλυση αλγορίθμου . . . 39
5.3.1 Απόδειξη ορθότητας . . . 39
5.3.2 Απόδειξη πολυπλοκότητας . . . 45

6 Καθαρισμός, φάση 2 47
6.1 Ο αλγόριθμος preorderCleaning . . . 47
6.2 Παράδειγμα εκτέλεσης αλγορίθμου . . . 47
6.3 Ανάλυση αλγορίθμου . . . 50
6.3.1 Απόδειξη ορθότητας . . . 50
6.3.2 Απόδειξη πολυπλοκότητας . . . 57
6.4 Άλλα 4 σημαντικά λήμματα . . . 57

7 Επεξήγηση κώδικα παραρτήματος 59
7.1 Το αρχείο BST.h . . . 59
7.2 Το αρχείο BST.c . . . 60
7.3 Το αρχείο placement.h . . . 62
7.4 Το αρχείο placement.c . . . 62
7.5 Το αρχείο cleaning.h . . . 65
7.6 Το αρχείο cleaning.c . . . 65

Βιβλιογραφία 67

Α΄ Υλοποίηση του αλγορίθμου μας σε C 71
Α΄.1 Ο κώδικας . . . 71
Α΄.1.1 Το αρχείο BST.h . . . 71
Α΄.1.2 Το αρχείο BST.c . . . 72
Α΄.1.3 Το αρχείο placement.h . . . 75
Α΄.1.4 Το αρχείο placement.c . . . 76
Α΄.1.5 Το αρχείο cleaning.h . . . 78
Α΄.1.6 Το αρχείο cleaning.c . . . 78
Α΄.2 Παράδειγμα εκτέλεσης κώδικα . . . 81

Κεφάλαιο 1

Εισαγωγή

Η πρώτη μαθηματική διατύπωση του Graph Searching δόθηκε το 1976 από τον Torrence Parsons [19] (βλέπε επίσης [20]). Ο Parsons εμπνεύστηκε την ιδέα από ένα προγενέστερο άρθρο του R. Breisch [6] που εμφανίστηκε το 1967. Στο άρθρο του ο Breisch πραγματεύεται την εύρεση ενός εξερευνητή που έχει χαθεί σε ένα πολύπλοκο σύστημα σκοτεινών σπηλαίων. Το ερώτημα που θέτει είναι το εξής: ποιος είναι ο ελάχιστος αριθμός από ερευνητές που μπορούν να εξέλθουν από τη σπηλιά με μία στρατηγική η οποία θα τους επιτρέπει να σώσουν τον εξερευνητή ανεξάρτητα από τις κινήσεις του; Ειδικότερα, η στρατηγική θα πρέπει να είναι επιτυχής ακόμα κι αν ο εξερευνητής σκοπίμως αποφεύγει τη συνάντησή του με τους ερευνητές.
Το 1993 οι Seymour και Thomas [22] όρισαν το Graph Searching ως ένα παιχνίδι δύο παικτών. Ο πρώτος παίκτης διαχειρίζεται μία οντότητα, η οποία καλείται φυγάς (fugitive) και κρύβεται μέσα σε ένα δίκτυο. Ο δεύτερος παίκτης διαχειρίζεται ένα πλήθος από διώκτες αυτής της οντότητας, τους οποίους καλούμε ερευνητές (searchers). Σκοπός των ερευνητών είναι να συλλάβουν το φυγά ντετερμινιστικά και του φυγά να αποφεύγει τη σύλληψη επ' αόριστον. Υπέθεσαν ότι ο φυγάς είναι πανίσχυρος υπό την έννοια ότι είναι αόρατος από τους ερευνητές, μονίμως ενήμερος για τη θέση τους στο δίκτυο, ενώ τρέχει και με απεριόριστη ταχύτητα. Τελική απόρροια όλων αυτών των υποθέσεων είναι ότι ο φυγάς μπορεί να αποφύγει τη σύλληψη εάν ένας δρόμος διαφυγής είναι διαθέσιμος. Ως εκ τούτου, από την πλευρά των ερευνητών μπορούμε να σκεφτούμε την αποστολή τους ως μια προσπάθεια αποκλεισμού των δρόμων διαφυγής.
Αυτού του είδους ο φυγάς μπορεί να παρομοιαστεί με ένα δηλητηριώδες αέριο που κυκλοφορεί στο δίκτυο. Η προσέγγιση αυτή απαλείφει το φυγά από το Graph Searching και τον αντικαθιστά με το μολυσμένο σύνολο. Έτσι, μπορούμε πλέον να αντιληφθούμε το Graph Searching ως ένα παιχνίδι για έναν παίκτη, ο οποίος διαχειρίζεται όλους του ερευνητές. Αρχικά, όλος ο γράφος είναι μολυσμένος. Σκοπός του παιχνιδιού είναι να καθαρίσουμε όλο το δίκτυο, χρησιμοποιώντας έναν ή περισσότερους ερευνητές. Αυτού του είδους η προσέγγιση δόθηκε για πρώτη φορά αρκετά χρόνια πριν από τον ορισμό των Seymour και Thomas, και συγκεκριμένα το 1978 από τον Parsons [19, 20]. Ο τυπικός ορισμός του Graph Searching που θα χρησιμοποιήσουμε στο κείμενο αυτό, ακολουθεί το πνεύμα του ορισμού του Parsons [19, 20].

1.1 Τυπικός ορισμός του παιχνιδιού

Κάθε δίκτυο, θα αναπαρίσταται στο θεωρητικό μας μοντέλο από ένα γράφημα G=(V,E). Αρχικά, ολόκληρο το γράφημα είναι μολυσμένο. Σκοπός του παιχνιδιού είναι να καθαρίσουμε όλες τις ακμές του γραφήματος, χρησιμοποιώντας έναν ή περισσότερους ερευνητές. Μία κίνηση ενός ερευνητή μπορεί να είναι μία από τις ακόλουθες: (α΄) *τοποθέτηση σε κόμβο u∈V*, (β΄) *αφαίρεση από κόμβο u∈V*, και (γ΄) *ολίσθηση από έναν κόμβο u∈V σε έναν γειτονικό του κόμβο v∈V μέσω της ακμής uv*. Μία *στρατηγική έρευνας* είναι μία πεπερασμένη ακολουθία από κινήσεις. Για ένα γράφημα G, μία στρατηγική έρευνας που εφαρμόζεται σε αυτό είναι *νικηφόρα*, αν τελικά καθαρίζει όλες τις ακμές του G.
Γενικά, μία ακμή e∈E(G) μπορεί να καθαριστεί με 2 τρόπους: (α΄) *όταν ένας ερευνητής ολισθήσει κατά μήκος της e*, και (β΄) *όταν και τα δύο άκρα της e καταλαμβάνονται ταυτόχρονα από ερευνητή*. Παρ' όλα αυτά, αν υπάρχει μονοπάτι, ελεύθερο από ερευνητές, ανάμεσα σε μία καθαρή και μία μολυσμένη ακμή, τότε η καθαρή ακμή *επαναμολύνεται (recontminated)*. Προφανώς, κάθε γράφημα G, n κόμβων, μπορεί σίγουρα να καθαριστεί χρησιμοποιώντας n ερευνητές, απλά τοποθετώντας έναν ερευνητή σε κάθε κόμβο του γραφήματος. Από την άλλη, όπως εύκολα γίνεται αντιληπτό, στα περισσότερα πρακτικά προβλήματα αυτό που στοιχίζει περισσότερο είναι ο αριθμός των ερευνητών που χρησιμοποιούνται από μια στρατηγική έρευνας. Το ζητούμενο λοιπόν είναι, δοθέντος ενός γράφου G, να υπολογιστεί μία στρατηγική έρευνας που να ελαχιστοποιεί τον αριθμό των ερευνητών που απαιτούνται για να καθαριστεί ο G. Αυτός ο αριθμός καλείται *search number* του γράφου G.

1.2 Παραλλαγές

Στο original graph searching, το οποίο στη βιβλιογραφία συναντάται ως *edge graph searching*, οι ερευνητές μπορούν να τοποθετούνται σε κόμβους του γραφήματος, να αφαιρούνται από τους κόμβους του γρφήματος ή να ολισθαίνουν κατά μήκος των ακμών του, ενώ κάθε ακμή του γραφήματος καθαρίζεται όταν ένας ερευνητής ολισθήσει κατά μήκος αυτής. Το αντίστοιχο search number, *es(G)*, ονομάζεται *edge search number*. Για παράδειγμα, για ένα μονοπάτι P, n>1 κόμβων, es(P)=1, ενώ η αντίστοιχη στρατηγική καθαρισμού περιλαμβάνει την τοποθέτηση ενός ερευνητή στο ένα άκρο του P και την ολίσθηση αυτού μέχρι το άλλο άκρο του P.
Η πρώτη παραλλαγή δόθηκε το 1985 από τους Λευτέρη Κυρούση και Χρήστο Παπαδημητρίου, οι οποίοι όρισαν το *node graph searching* [13]. Σύμφωνα με αυτήν, οι ερευνητές μπορούν να τοποθετούνται σε κόμβους του γραφήματος ή να αφαιρούνται από αυτούς, ενώ η ολίσθηση κατά μήκος των ακμών του γραφήματος απαγορεύεται. Εδώ, μία μολυσμένη ακμή καθαρίζεται όταν και τα δυο της άκρα καταλαμβάνονται ταυτόχρονα από ερευνητή. Το search number που αντστοιχεί σε αυτήν την παραλλαγή είναι γνωστό ως *node search number* και συμβολίζεται με *ns(G)*. Για παράδειγμα, για ένα μονοπάτι P, που αποτελείται από n κόμβους, με n>1, ns(P)=2. Ειδικότερα, ένας ερευνητής τοποθετείται αρχικά στη v_1. Τότε, για i=2 έως n, ένας ερευνητής τοποθετείται στη v_i, ενώ αφαιρείται ο ερευνητής από τη v_{i-1}. Σε αυτήν την παραλλαγή, ένας ερευνητής δεν αρκεί για να καθαριστεί το γράφημα, αφού κάθε φορά που αυτός θα αφαιρείται από το γράφημα, τότε ολόκληρο το γράφημα θα

επαναμολύνεται.
Μία άλλη παραλλαγή, γνωστή ως *mixed graph searching*, δόθηκε 6 χρόνια αργότερα, από τους Daniel Bienstock και Seymour [4]. Σε αυτήν την παραλλαγή, επιτρέπεται η τοποθέτηση ενός ερευνητή σε έναν κόμβο του γραφήματος, η αφαίρεση ενός ερευνητή από έναν κόμβο του γραφήματος καθώς και η ολίσθηση ενός ερευνητή κατά μήκος μιας ακμής του γραφήματος, ενώ μία ακμή καθαρίζεται όταν ένας ερευνητής ολισθήσει κατά μήκος αυτής ή όταν και τα δυο της άκρα καταλαμβάνονται ταυτόχρονα από ερευνητή. Το αντίστοιχο search number καλείται *mixed search number* και συμβολίζεται με *s(G)*. Ένα παράδειγμα mixed στρατηγικής αποτελεί η προαναφερθείσα edge στρατηγική.
Το node search number ενός γράφου G συνδέεται με το pathwidth του G [21], με την ακόλουθη ισότητα: mns(G)=pw(G)+1 [13, 4]. Αφού ο υπολογισμός του pathwidth είναι NP-hard σε πολλές κλάσεις γραφημάτων (π.χ., [12, 17]), θα θέλαμε για όσο το δυνατόν περισσότερες από αυτές τις κλάσεις, έναν αλγόριθμο που να υπολογίζει κάποιο από τα edge, node ή mixed search numbers σε πολυωνυμικό χρόνο. Τότε, θα είχαμε και έναν πολυωνυμικού χρόνου αλγόριθμο που θα υπολόγιζε το pathwidth για κάθε γράφημα που ανήκει στην αντίστοιχη κλάση. Δυστυχώς, δε γνωρίζουμε καμία οικογένεια γραφημάτων για την οποία η πολυπλοκότητα του pathwidth να διαφέρει από αυτήν του search number.
Πρόσφατα, οι Blin κ.α. εισήγαγαν μία νέα παραλλαγή που ονομάζεται *exclusive graph searching* [5]. Αυτή η στρατηγική περιλαμβάνει αρχικά την τοποθέτηση k ερευνητών σε k διαφορετικούς κόμβους ενός γραφήματος G=(V,E). Σε κάθε βήμα, ο ερευνητής ενός κόμβου u∈V μπορεί να ολισθήσει κατά μήκος μιας ακμής uv∈ E αν και μόνο αν ο κόμβος u δεν καταλαμβάνεται ήδη από άλλον ερευνητή. Έπεται ότι η μεταπήδηση ενός ερευνητή από έναν κόμβο u∈V σε έναν άλλον μη γειτονικό του, απαγορεύεται. Το search number που αντιστοιχεί στην τελευταία παραλλαγή είναι γνωστό ως *exclusive search number*, και συμβολίζεται με *xs(G)*. Για παράδειγμα, για έναν αστέρα S, n>2 κόμβων, xs(S) =n-2. Ειδικότερα, n-2 ερευνητές τοποθετούνται αρχικά σε n-2 διαφορετικά φύλλα, δηλαδή σε όλα εκτός από ένα, έστω v αυτό. Έπειτα, ένας οποιοσδήποτε ερευνητής ολισθαίνει προς τον κεντρικό κόμβο του αστέρα. Τέλος, ο ερευνητής ο ερευνητής ολισθαίνει από τον κεντρικό κόμβο προς τον κόμβο v.
Για το exclusive graph searching βρέθηκαν οικογένειες γραφημάτων για τις οποίες η πολυπλοκότητα του pathwidth διαφέρει από εκείνη του search number [15]. Δυστυχώς, όμως, σε αυτήν την περίπτωση δε γνωρίζουμε μία σχέση που να συνδέει το pathwidth με το exclusive search number.

1.3 Ιδιότητες του Graph Searching

Ένα επιθυμητό χρακτηριστικό μιας στρατηγικής έρευνας είναι το σύνολο των καθαρών ακμών να μη μειώνεται ή, ισοδύναμα, το σύνολο των μολυσμένων ακμών να μην αυξάνεται. Με άλλα λόγια, από τη στιγμή που μία μολυσμένη ακμή καθίσταται καθαρή να μην επαναμολύνεται. Μία edge (node, mixed, exclusive) στρατηγική που ενσωματώνει το παραπάνω χαρακτηριστικό καλείται *monotone edge (node, mixed, exclusive) στρατηγική* [4, 14], ενώ το αντίστοιχο search number, *mes(G) (mns(G), ms(G), mxs(G))*, καλείται *monotone edge (node, mixed, exclusive) search number*.

Ένα άλλο χαρακτηριστικό που μπορούμε να προσδώσουμε σε μία στρατηγική είναι το καθαρό μέρος του γράφου να παραμένει συνεκτικό καθόλη τη διάρκεια της στρατηγικής. Αυτού του είδους το graph searching ορίστηκε από την Barriere κ.α. [2, 3, 1] (βλεπε είσης [7, 8, 9, 10, 11, 18]). Μία edge (node, mixed, exclusive) στρατηγική που ικανοποιεί την παραπάνω ιδιότητα καλείται *connected edge (node, mixed, exclusive) στρατηγική*, ενώ το αντίστοιχο search number, *ces(G) (cns(G), cs(G), cxs(G))*, καλειται *connected edge (node, mixed, exclusive) search number*.
Μία τελευταία ιδιότητα που μπορεί να εμπεριέχεται σε μια στρατηγική έρευνας απαγορεύει τη μεταπήδηση ενός ερευνητή από έναν κόμβο $u \in V$ σε έναν μη γειτονικό του. Μία edge (node, mixed) στρατηγική έρευνας που υπακούει στην παραπάνω ιδιότητα καλείται *internal edge (node, mixed) στρατηγική* [1], ενώ το αντίστοιχο search number, *ies(G) (ins(G), is(G))*, καλειται *internal edge (node, mixed) search number*. Εξάλλου, μία exclusive στρατηγική είναι εξ' ορισμού internal.

1.4 Σχετική δουλειά

Τα edge, node και mixed search numbers διαφέρουν, όχι περισσότερο από μία σταθερά, το ένα από το άλλο. Ειδικότερα, για κάθε γράφημα G ισχύει, $ns(G)-1 \leq es(G) \leq ns(G)+1$ [13], και, $s(G) \leq ns(G) \leq s(G)+1$ [4]. Εξάλλου, δεδομένης μίας βέλτιστης edge στρατηγικής, μία βέλτιστη node στρατηγική μπορεί να κατασκευαστεί, και αντίστροφα [13], ενώ, δεδομένης μίας βέλτιστης mixed στρατηγικής, μία βέλτιστη edge (node) στρατηγική μπορεί να κατασκευαστεί, και αντίστροφα [4].
Το πρόβλημα απόφασης που αντιστοιχεί στον υπολογισμό του edge (node, mixed) search number είναι NP-complete για κάθε γράφημα G [16] ([13], [4]), ενώ κάθε ένα από τα edge, node, mixed graph searching είναι monotone. Δηλαδή, για κάθε γράφημα G, υπάρχει μία monotone edge (node, mixed) στρατηγική που καθαρίζει το G χρησιμοποιώντας es(G) (ns(G), s(G)) ερευνητές [14] ([13], [4]).
Το connected graph searching, στη γενική περίπτωση, δεν είναι monotone [23]. Δηλαδή, υπάρχουν γραφήματα G για τα οποία, $cs(G)<mcs(G)$. Αντίθετα, είναι monotone, για κάθε δέντρο T.
Exclusive Graph Searching. Αν και κάθε exclusive στρατηγική είναι mixed (κατά συνέπεια $s(G) \leq xs(G)$ για κάθε γράφημα G), ωστόσο τα αποτελέσματα στο [5] δείχνουν ότι το exclusive graph searching φαίνεται να συμπεριφέρεται διαφορετικά από το edge (node, mixed) graph searching αφου (α΄) δεν είναι monotone (β΄) υπάρχουν γραφήματα G και H τέτοια ώστε το H είναι υπογράφημα του G και $xs(H)>xs(G)$. Πράγματι, σε ένα γράφημα G, το exclusive search number του G μπορεί να διαφέρει εκθετικά από το edge (node, mixed) search number του G. Έστω, για παράδειγμα, S, ένας αστέρας με έναν κεντρικό κόμβο c και n-1 φύλλα. Στα edge, node και mixed graph searching ένας ερευνητής μπορεί να βρίσκεται στον κόμβο c και ένας δεύτερος να καθαρίσει όλες τις ακμές του αστέρα είτε μεταπηδώντας από φύλλο σε φύλλο, είτε ολισθαίνοντας από φύλλο σε φύλλο περνώντας κάθε φορά από τον κεντρικό κόμβο c. Στο exclusive graph searching, ωστόσο, τέτοιες τακτικές δεν επιτρέπονται, αφού από τη μία απαγορεύεται η μεταπήδηση, ενώ από την άλλη απαγορεύεται δύο ερευνητές να βρίσκονται στον ίδιο κόμβο την ίδια χρονική στιγμή. Έτσι λοιπόν,

$es(S)=ns(S)=s(S)=2$, ενώ $xs(S)=n-2$.

Exclusive Graph Searching για δεντρικές τοπολογίες. Οι Blin κ.α. [5] απέδειξαν ότι (α΄) το exclusive graph searching δεν είναι monotone ούτε καν στα δέντρα (β΄) για κάθε δέντρο T και για κάθε υποδέντρο R του T, $xs(R)<xs(T)$ (γ΄) υπάρχει ένας πολυωνυμικού χρόνου αλγόριθμος ο οποίος υπολογίζει το xs(T) και μία αντίστοιχη exclusive στρατηγική, για κάθε δέντρο T. Απέδειξαν επίσης ότι, $ns(G)-1\leq xs(G)\leq(\Delta-1)\cdot ns(G)$, για κάθε γράφημα G με μέγιστο βαθμό Δ, ενώ αν $\Delta\leq 3$, $ns(G)-1\leq xs(G)\leq ns(G)$.

Κεφάλαιο 2

Επισκόπηση αλγορίθμου

Ο αλγόριθμος δέχεται ως είσοδο ένα τυχαίο δυαδικό δέντρο T, όλες οι ακμές του οποίου είναι μολυσμένες, και τοποθετεί ερευνητές σε κάποιους από τους κόμβους του T έτσι ώστε, αν αυτοί εκτελέσουν μία σείρα από καθορισμένες κινήσεις, καταφέρνουν να καθαρίσουν όλες τις ακμές του δέντρου.

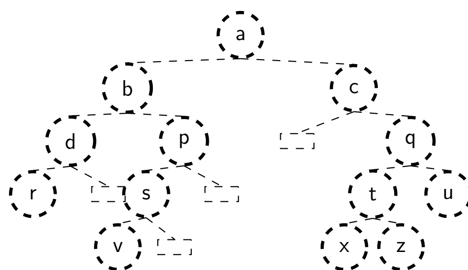
2.1 Η διαδικασία της επισήμανσης

Αν θεωρήσουμε όλους τους κόμβους του δέντρου μη επισημασμένους, τότε:

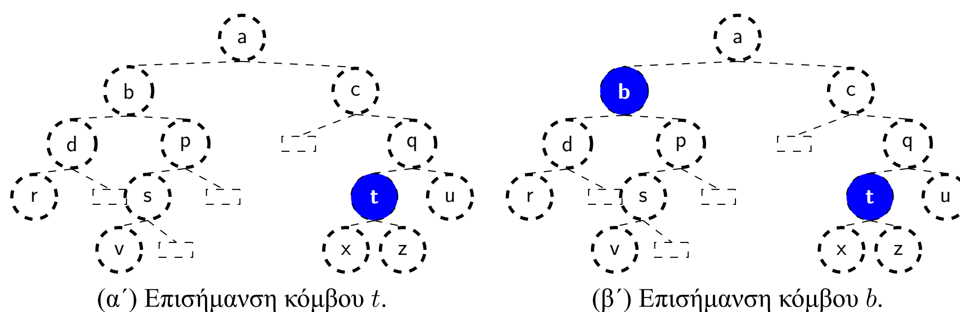
- Επισημαίνουμε πάντα τη ρίζα του δέντρου.
- Κάθε εσωτερικός κόμβος του δέντρου, για τον οποίο έχουμε ήδη επεξεργαστεί το αριστερό και το δεξί του παιδί, επισημαίνεται αν και μόνο αν ο κόμβος έχει ακριβώς 2 μη επισημασμένα παιδιά.

Έστω για παράδειγμα το δέντρο του Σχήματος 2.1. Αν v, x, z, u, t, s, r, d, p, q, c, b, a η σειρά με την οποία ο αλγόριθμος επεξεργάζεται τους κόμβους, τότε έχουμε:

- Οι κόμβοι v, x, z, u είναι φύλλα του δέντρου και κατά συνέπεια δεν έχουν παιδιά.
- Ο κόμβος t έχει 2 μη επισημασμένα παιδιά. Κατά συνέπεια, ο κόμβος t επισημαίνεται (Σχήμα 2.2α΄).
- Οι κόμβοι s, r, d, p, q και c, εκτός του r που δεν έχει μη επισημασμένα παιδιά, οι υπόλοιποι έχουν από 1 μη επισημασμένο παιδί.
- Ο κόμβος b έχει 2 μη επισημασμένα παιδιά. Κατά συνέπεια, ο κόμβος b επισημαίνεται (Σχήμα 2.2β΄).
- Η ρίζα a του δέντρου, επίσης επισημαίνεται (Σχήμα 2.2γ΄).

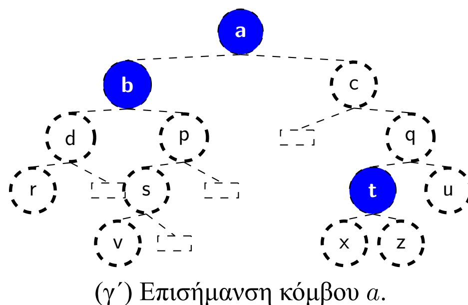


Σχήμα 2.1: Τυχαίο δυαδικό δέντρο T, όλες οι ακμές του οποίου είναι μολυσμένες.



(α΄) Επισήμανση κόμβου t.

(β΄) Επισήμανση κόμβου b.



(γ΄) Επισήμανση κόμβου a.

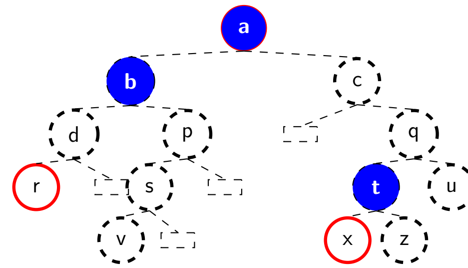
Σχήμα 2.2: Επισήμανση κόμβων.

2.2 Η διαδικασία της τοποθέτησης

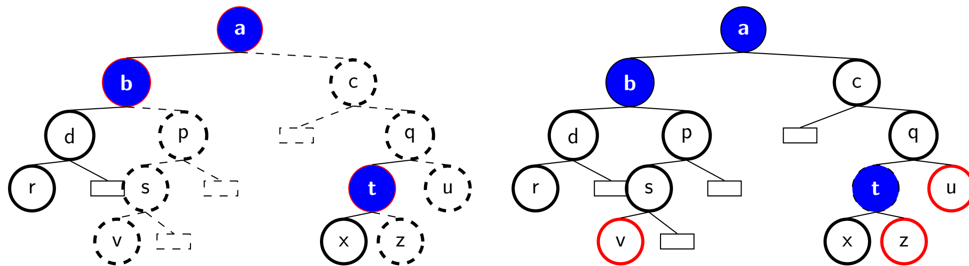
Εδώ, τοποθετούμε από έναν ερευνητή σε κάθε εσωτερικό κόμβο του δέντρου που ανήκει στο αριστερό υποδέντρο του πλησιέστερου επισημασμένου προγόνου του και ο οποίος δεν έχει μη επισημασμένα παιδιά ή στη ρίζα του δέντρου αν αυτή δεν έχει αριστερό μη επισημασμένο παιδί (Σχήμα 2.3).

2.3 Η διαδικασία του καθαρισμού, φάση 1

Κάθε ερευνητής απομακρύνεται από τον κόμβο στον οποίο μόλις εισήχθει και φτάνει στον πλησιέστερο επισημασμένο πρόγονο, ολισθαίνοντας κατά μήκος όλων των ακμών που υπάρχουν μεταξύ της αρχικής και της τελικής του θέσης. Κατά συνέπεια, κάθε τέτοια ακμή καθαρίζει. Ο ερευνητής που κινείται 1ος είναι εκείνος που για να φτάσει στον πλησιέστερο



Σχήμα 2.3: Εισαγωγή ερευνητών στους κόμβους r, x και a.



(α΄) Καθαρισμός, φάση 1. Μετακίνηση των ερευνητών από τους κόμβους r, x στους κόμβους b, t αντίστοιχα.

(β΄) Καθαρισμός, φάση 2. Μετακίνηση των ερευνητών από τους κόμβους a, b και t στους κόμβους u, v και z αντίστοιχα.

Σχήμα 2.4: Καθαρισμός.

επισημασμένο πρόγονο θα πρέπει να διασχίσει τις περισσότερες ακμές. Αν 2 ή περισσότεροι ερευνητές χρειάζεται να διασίχισουν τον ίδιο αριθμό ακμών για να φτάσουν στον πλησιέστερο επισημασμένο πρόγονο, τότε η σειρά με την οποία θα κινηθούν είναι τυχαία. Στο δέντρο του Σχήματος 2.3, για παράδειγμα, ο ερευνητής του κόμβου r θα πρέπει να διασχίσει 2 ακμές, τις rd και db, για να φτάσει στον πλησιέστερο επισημασμένο πρόγονο b, ενώ ο ερευνητής του κόμβου x αν διασχίσει 1 μόνο ακμή, την ακμή xt, θα φτάσει στον πλησιέστερο επισημασμένο πρόγονο t. Κατά συνέπεια, κινείται 1ος ο ερευνητής του κόμβου r και 2ος ο ερευνητής του κόμβου x (Σχήμα 2.4α΄). Ο ερευνητής του κόμβου a, μιας και βρίσκεται ήδη σε επισημασμένο κόμβο, παραμένει στη θέση του.

2.4 Η διαδικασία του καθαρισμού, φάση 2

Έπειτα, κάθε ερευνητής απομακρύνεται από τον επισημασμένο κόμβο στον οποίο βρίσκεται και φτάνει σε εκείνον τον μη επισημασμένο απόγονο, ο οποίος δεν έχει μη επισημασμένα παιδιά και ανήκει στο δεξιό υποδέντρο του πλησιέστερου επισημασμένου πρόγονου του, ο οποίος είναι εκείνος ο κόμβος από τον οποίο ξεκίνησε την καθοδική του πορεία ο ερευνητής. Για να το πετύχει αυτό, κάθε ερυνητής ολισθαίνει κατά μήκος όλων των ακμών που υπάρχουν μεταξύ της αρχικής και της τελικής του θέσης, οι οποίες κατά συνέπεια καθαρίζουν. Ο ερευνητής που κινείται 1ος είναι εκείνος που βρίσκεται πιο κοντά στη ρίζα του δέντρου. Αν 2 ή περισσότεροι ερευνητές απέχουν τον ίδιο αριθμό ακμών από τη ρίζα, τότε

η σειρά με την οποία θα κινηθούν είναι τυχαία. Στο δέντρο του Σχήματος 2.4α΄, για παράδειγμα, ο ερευνητής του κόμβου a βρίσκεται στη ρίζα του δέντρου, αυτός του κόμβου t απέχει 3 ακμές από τη ρίζα, ενώ ο ερευνητής του κόμβου b απέχει μόλις 1 ακμή από τη ρίζα. Κατά συνέπεια, κινείται 1ος ο ερευνητής του κόμβου a, 2ος ο ερευνητής του κόμβου b και 3ος ο ερευνητής του κόμβου t.(Σχήμα 2.4β΄).

Κεφάλαιο 3

Επισήμανση κόμβων

Η διαδικασία περιγράφεται από τον αλγόριθμο $postorderLabeling$. Η αρχική κλήση είναι $postorderLabeling(root)$, ενώ πριν αυτό συμβεί όλες οι ακμές του δέντρου είναι μολυσμένες και όλοι οι κόμβοι του μη επισημασμένοι.

3.1 Ο αλγόριθμος postorderLabeling

Είσοδος: ο κόμβος u του οποίου το υποδέντρο T_u θα διαπεράσουμε, 0
Έξοδος: το πλήθος των επισημασμένων κόμβων του T_u

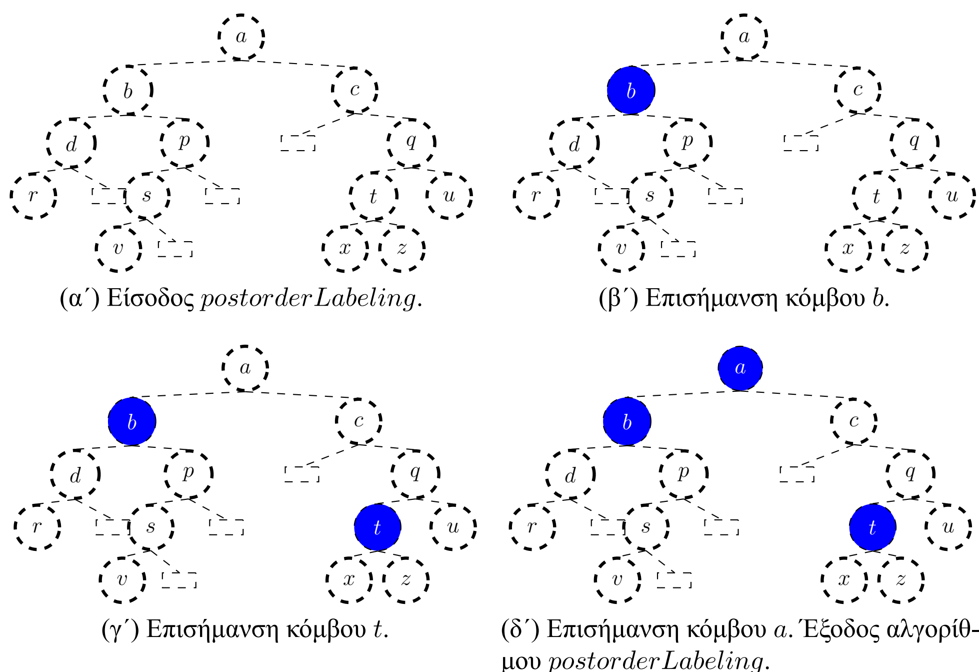
```
 1: function postorderLabeling(κόμβος u, ακέραιος nof_lodes)
 2:     if ο u υπάρχει then
 3:         nof_lnodes ← postorderLabeling(u, nof_lnodes)
 4:         nof_lnodes ← postorderLabeling(u, nof_lnodes)
 5:         if ο u δεν έχει πατέρα or ο u έχει 2 μη επισημασμένα παιδιά then
 6:             επισήμανε τον u
 7:             nof_lnodes ← nof_lnodes + 1
 8:         end if
 9:         return nof_lnodes
10:     end if
11: end function
```

Algorithm 1: Επισήμανση Κόμβων.

3.2 Παράδειγμα Εκτέλεσης αλγορίθμου

Για το τυχαίο δυαδικό δέντρο του Σχήματος 3.1α΄, εφαρμόζουμε τον αλγόριθμο $postorderLabeling$:

- *Επίσκεψη στους κόμβους r, d, v, s και p* (Σχήμα 3.1α΄), οι οποίοι είναι όλοι τους εσωτερικοί κόμβοι. Από αυτούς, οι κόμβοι r και v δεν έχουν παιδιά, ενώ οι υπόλοιποι τρεις έχουν από ένα παιδί ο καθένας.



(α΄) Είσοδος $postorderLabeling$.

(β΄) Επισήμανση κόμβου b.

(γ΄) Επισήμανση κόμβου t.

(δ΄) Επισήμανση κόμβου a. Έξοδος αλγορίθμου $postorderLabeling$.

Σχήμα 3.1: Στιγμιότυπο αλγορίθμου $postorderLabeling$.

- *Επίσκεψη στον κόμβο* b (Σχήμα 3.1α΄), ο οποίος είναι ένας εσωτερικός κόμβος που έχει 2 μη επισημασμένα παιδιά (κόμβοι d και p). Κατά συνέπεια, Σχήμα 3.1β΄, ο κόμβος b επισημαίνεται.

- *Επίσκεψη στους κόμβους* x *και* z (Σχήμα 3.1β΄). Είναι και οι δύο εσωτερικοί κόμβοι, ενώ κανένας τους δεν έχει παιδιά.

- *Επίσκεψη στον κόμβο* t (Σχήμα 3.1β΄), ο οποίος είναι ένας εσωτερικός κόμβος που έχει 2 μη επισημασμένα παιδιά (κόμβοι x και z). Κατά συνέπεια, Σχήμα 3.1γ΄, ο κόμβος t επισημαίνεται.

- *Επίσκεψη στους κόμβους* u*,* q *και* c (Σχήμα 3.1γ΄), οι οποίοι είναι όλοι τους εσωτερικοί κόμβοι. Από αυτούς, ο κόμβος u δεν έχει παιδιά, ενώ οι υπόλοιποι έχουν από ένα παιδί ο καθένας.

- *Επίσκεψη στη ρίζα* a *του δέντρου* (Σχήμα 3.1γ΄), η οποία επισημαίνεται (Σχήμα 3.1δ΄).

3.3 Ανάλυση Αλγορίθμου

3.3.1 Ανάλυση ορθότητας

Λήμμα 3.3.1. *Ο αλγόριθμος* $postorderLabeling$ *επισκέπτεται με* $postorder$ *όλους τους κόμβους του δέντρου. Ένας κόμβος επισημαίνεται από τον αλγόριθμο αν και μόνο αν είναι η ρίζα*

του δέντρου ή έχει 2 μη επισημασμένα παιδιά.

Απόδειξη.

- Αν *ο u δεν υπάρχει*, τότε η συνθήκη ελέγχου της 2ης γραμμής του αλγορίθμου είναι ψευδής.
- Αν *ο u υπάρχει*, τότε η συνθήκη ελέγχου της 2ης γραμμής του αλγορίθμου είναι αληθής, οπότε εκτελούνται η 3η, η 4η και η 5η γραμμή του αλγορίθμου. Στις γραμμές 3 και 4 ο αλγόριθμος καλεί αναδρομικά τον εαυτό του με είσοδο το αριστερό και το δεξί του παιδί αντίστοιχα, ενώ στη γραμμή 5 λαμβάνει χώρα μία συνθήκη ελέγχου η οποία,
  - αν *ο u είναι η ρίζα του δέντρου*, είναι αληθής. Τότε εκτελούνται η 6η και η 7η γραμμή του αλγορίθμου, όπου επισημαίνεται ο κόμβος u (γραμμή 6) και αυξάνεται κατά 1 η τιμή της μεταβλητής που σώζει το πλήθος των επισημασμένων κόμβων του δέντρου (γραμμή 7). Έπειτα εκτελείται η 9η γραμμή του αλγορίθμου.
  - αν *u είναι κάποιος από τους εσωτερικούς κόμβος του δέντρου*, ο οποίος
    * *έχει 2 μη επισημασμένα παιδιά*, είναι αληθής. Τότε εκτελούνται η 6η και η 7η γραμμή του αλγορίθμου, όπου επισημαίνεται ο κόμβος u (γραμμή 6) και αυξάνεται κατά 1 η τιμή της μεταβλητής που σώζει το πλήθος των επισημασμένων κόμβων του δέντρου (γραμμή 7). Έπειτα εκτελείται η 9η γραμμή του αλγορίθμου.
    * *δεν έχει 2 μη επισημασμένα παιδιά*, είναι ψευδής. Έπειτα εκτελείται η 9η γραμμή του αλγορίθμου.

Η γραμμή 9, τέλος, επιστρέφει το πλήθος των επισημασμένων κόμβων του δέντρου. □

3.3.2 Απόδειξη πολυπλοκότητας

Λήμμα 3.3.2. *Η κλήση $postorderLabeling(u)$, όπου u η ρίζα ενός τυχαίου δυαδικού δέντρου T με n κόμβους, απαιτεί χρόνο $\Theta(n)$.*

Απόδειξη. Αρκεί να αποδείξουμε ότι ο χρόνος d που χρειάζεται ο αλγόριθμος μας για να επεξεργαστεί 1 μόνο κόμβο (γραμμές $5-9$) είναι σταθερός και ανεξάρτητος του μεγέθους της εισόδου n. Ο παρακάτω πίνακας συνοψίζει το κόστος εκτελέσεως και το πλήθος εκτελέσεων κάθε γραμμής κατά τη διάρκεια της επεξεργασίας ενός κόμβου u από τον αλγόριθμο $postorderLabeling$:

Γραμμή	*Κόστος*	*Αριθμός εκτελέσεων*
5	c_5	1
6	c_6	≤ 1
7	c_7	≤ 1
9	c_9	1

$$d \leq c_5 + c_6 + c_7 + c_9$$

Αυτός ο χρόνος εκτέλεσης μπορεί να γραφεί στη μορφή a όπου η σταθερά a εξαρτάται από τα διάφορα κόστη c_i των εντολών. Επομένως, είναι μια σταθερή συνάρτηση του n. $\square$

Κεφάλαιο 4

Εισαγωγή ερευνητών στο δυαδικό δέντρο

Η διαδικασία περιγράφεται από τον αλγόριθμο $putSearchers$. Η αρχική κλήση είναι $putSearchers(root)$, ενώ πριν αυτό συμβεί θα πρέπει να έχει κληθεί ο αλγόριθμος $postorderLabeling$.

4.1 Ο αλγόριθμος putSearchers

Είσοδος: ο κόμβος u του οποίου το υποδέντρο T_u θα διαπεράσουμε, -1, 0
Έξοδος: το πλήθος των ερευνητών που θα καθαρίσουν το δέντρο

1: **function** $put_searchers$(κόμβος u, ακέραιος $isleft$, ακέραιος $nof_searchers$)
2:     **if** ο u υπάρχει **then**
3:         **if** ο u είναι επισημασμένος **then**
4:             $nof_searchers \leftarrow putSearchers$(αριστερό παιδί του u, 1, $nof_searchers$)
5:             $nof_searchers \leftarrow putSearchers$(δεξί παιδί του u, 0, $nof_searchers$)
6:         **else**
7:             $nof_searchers \leftarrow putSearchers$(αριστερό παιδί u, $isleft$, $nof_searchers$)
8:             $nof_searchers \leftarrow putSearchers$(δεξί παιδί του u, $isleft$, $nof_searchers$)
9:         **end if**
10:         **if** ο u είναι η ρίζα του δέντρου **then**
11:             **if** ο u δεν έχει αριστερό παιδί ή έχει αριστερό επισημασμένο παιδί **then**
12:                 βάλε ερευνητή στον κόμβο u
13:                 $nof_searchers \leftarrow nof_searchers + 1$
14:             **end if**
15:         **else**
16:             **if** ο u δεν έχει μη επισημασμένα παιδιά **then**
17:                 **if** $isleft = 1$ **then**

```
18:                 βάλε ερευνητή στον u
19:                 nof_searchers ← nof_searchers + 1
20:             end if
21:         end if
22:     end if
23:     return nof_searchers
24:   end if
25: end function
```

Algorithm 2: Είσαγωγή ερευνητών στο δυαδικό δέντρο.

4.2 Παράδειγμα εκτέλεσης αλγορίθμου

Για το τυχαίο δυαδικό δέντρο του Σχήματος 4.1α΄ εφαρμόζουμε τον αλγόριθμο $putSearchers$:

- 1*η επίσκεψη στον κόμβο* a. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για τον κόμβο b, δηλαδή για το αριστερό παιδί ενός επισημασμένου κόμβου. Κατά συνέπεια, η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου b ισούται με 1.

- 1*η επίσκεψη στον κόμβο* b. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για τον κόμβο d, δηλαδή για το αριστερό παιδί ενός επισημασμένου κόμβου. Κατά συνέπεια, η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου d ισούται με 1.

- 1*η επίσκεψη στον κόμβο* d. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για τον κόμβο r, δηλαδή για το παιδί ενός μη επισημασμένου κόμβου. Κατά συνέπεια, η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου r ισούται με την τιμή της τοπικής μεταβλητής $isleft$ του κόμβου d, δηλαδή με 1.

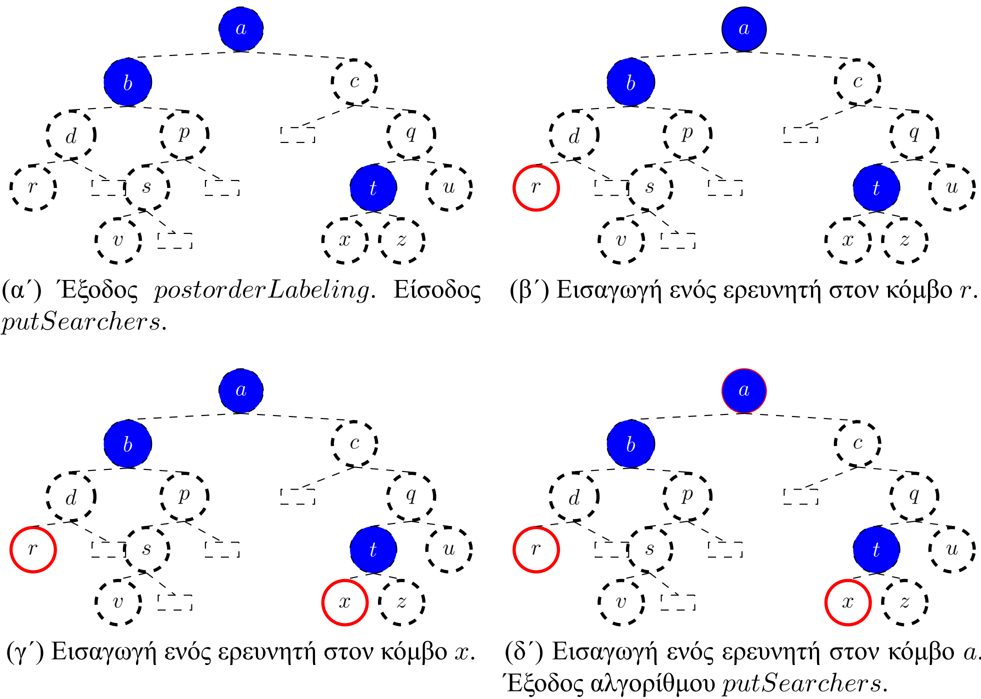
- 1*η επίσκεψη στον κόμβο* r. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για το αριστερό παιδί του κόμβου r, ο οποίος όμως δεν έχει αριστερό παιδί.

- 2*η επίσκεψη στον κόμβο* r. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για το δεξί παιδί του κόμβου r, ο οποίος όμως δεν έχει δεξί παιδί.

- 3*η επίσκεψη στον κόμβο* r. Ο αλγόριθμος επεξεργάζεται τον κόμβο r, ο οποίος είναι ένας εσωτερικός κόμβος που δεν έχει μη επισημασμένα παιδιά, ενώ η τιμή της τοπικής του μεταβλητής $isleft$ ισούται με 1. Κατά συνέπεια, Σχήμα 4.1β΄, ένας ερευνητής εισάγεται στον κόμβο r.

- 2*η επίσκεψη στον κόμβο* d. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για το δεξί παιδί του κόμβου d, ο οποίος όμως δεν έχει δεξί παιδί.

- 3*η επίσκεψη στον κόμβο* d. Ο αλγόριθμος επεξεργάζεται τον κόμβο d, ο οποίος έχει 1 μη επισημασμένο παιδί.



(α΄) Έξοδος $postorderLabeling$. Είσοδος $putSearchers$.

(β΄) Εισαγωγή ενός ερευνητή στον κόμβο r.

(γ΄) Εισαγωγή ενός ερευνητή στον κόμβο x.

(δ΄) Εισαγωγή ενός ερευνητή στον κόμβο a. Έξοδος αλγορίθμου $putSearchers$.

Σχήμα 4.1: Στιγμιότυπο αλγορίθμου $putSearchers$.

- *2η επίσκεψη στον κόμβο b.* Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για τον κόμβο p, δηλαδή για το δεξί παιδί ενός επισημασμένου κόμβου. Κατά συνέπεια, η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου p ισούται με 0.

- *1η επίσκεψη στον κόμβο p.* Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για τον κόμβο s, δηλαδή για το παιδί ενός μη επισημασμένου κόμβου. Κατά συνέπεια, η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου s ισούται με την τιμή της τοπικής μεταβλητής $isleft$ του κόμβου p, δηλαδή με 0.

- *1η επίσκεψη στον κόμβο s.* Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για τον κόμβο v, δηλαδή για το παιδί ενός μη επισημασμένου κόμβου. Κατά συνέπεια, η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου v ισούται με την τιμή της τοπικής μεταβλητής $isleft$ του κόμβου s, δηλαδή με 0.

- *1η επίσκεψη στον κόμβο v.* Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για το αριστερό παιδί του κόμβου v, ο οποίος όμως δεν έχει αριστερό παιδί.

- *2η επίσκεψη στον κόμβο v.* Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για το δεξί παιδί του κόμβου v, ο οποίος όμως δεν έχει δεξί παιδί.

- *3η επίσκεψη στον κόμβο v.* Ο αλγόριθμος επεξεργάζεται τον κόμβο v, ο οποίος αν και είναι ένας εσωτερικός κόμβος που δεν έχει μη επισημασμένα παιδιά, η τιμή της τοπικής του μεταβλητής $isleft$ ισούται με 0.

- 2*η επίσκεψη στον κόμβο* s. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για το δεξί παιδί του κόμβου s, ο οποίος όμως δεν έχει δεξί παιδί.
- 3*η επίσκεψη στον κόμβο* s. Ο αλγόριθμος επεξεργάζεται τον κόμβο s, ο οποίος έχει 1 μη επισημασμένο παιδί.
- 2*η επίσκεψη στον κόμβο* p. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για το δεξί παιδί του κόμβου p, ο οποίος όμως δεν έχει δεξί παιδί.
- 3*η επίσκεψη στον κόμβο* p. Ο αλγόριθμος επεξεργάζεται τον κόμβο p, ο οποίος έχει 1 μη επισημασμένο παιδί.
- 3*η επίσκεψη στον κόμβο* b. Ο αλγόριθμος επεξεργάζεται τον κόμβο b, ο οποίος έχει 2 μη επισημασμένα παιδιά.
- 2*η επίσκεψη στον κόμβο* a. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για τον κόμβο c, δηλαδή για το δεξί παιδί ενός επισημασμένου κόμβου. Κατά συνέπεια, η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου c ισούται με 0.
- 1*η επίσκεψη στον κόμβο* c. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για το αριστερό παιδί του κόμβου c, ο οποίος όμως δεν έχει αριστερό παιδί.
- 2*η επίσκεψη στον κόμβο* c. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για τον κόμβο q, δηλαδή για το παιδί ενός μη επισημασμένου κόμβου. Κατά συνέπεια, η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου q ισούται με την τιμή της τοπικής μεταβλητής $isleft$ του κόμβου c, δηλαδή με 0.
- 1*η επίσκεψη στον κόμβο* q. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για τον κόμβο t, δηλαδή για το παιδί ενός μη επισημασμένου κόμβου. Κατά συνέπεια, η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου t ισούται με την τιμή της τοπικής μεταβλητής $isleft$ του κόμβου q, δηλαδή με 0.
- 1*η επίσκεψη στον κόμβο* t. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για τον κόμβο x, δηλαδή για το αριστερό παιδί ενός επισημασμένου κόμβου. Κατά συνέπεια, η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου x ισούται με 1.
- 1*η επίσκεψη στον κόμβο* x. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για το αριστερό παιδί του κόμβου x, ο οποίος όμως δεν έχει αριστερό παιδί.
- 2*η επίσκεψη στον κόμβο* x. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για το δεξί παιδί του κόμβου x, ο οποίος όμως δεν έχει δεξί παιδί.
- 3*η επίσκεψη στον κόμβο* x. Ο αλγόριθμος επεξεργάζεται τον κόμβο x, ο οποίος είναι ένας εσωτερικός κόμβος που δεν έχει μη επισημασμένα παιδιά, ενώ η τιμή της τοπικής του μεταβλητής $isleft$ ισούται με 1. Κατά συνέπεια, Σχήμα 4.1γ΄, ένας ερευνητής εισάγεται στον κόμβο x.

- 2*η επίσκεψη στον κόμβο* t. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για τον κόμβο z, δηλαδή για το δεξί παιδί ενός επισημασμένου κόμβου. Κατά συνέπεια, η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου z ισούται με 0.

- 1*η επίσκεψη στον κόμβο* z. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για το αριστερό παιδί του κόμβου z, ο οποίος όμως δεν έχει αριστερό παιδί.

- 2*η επίσκεψη στον κόμβο* z. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για το δεξί παιδί του κόμβου z, ο οποίος όμως δεν έχει δεξί παιδί.

- 3*η επίσκεψη στον κόμβο* z. Ο αλγόριθμος επεξεργάζεται τον κόμβο z, ο οποίος αν και είναι ένας εσωτερικός κόμβος που δεν έχει μη επισημασμένα παιδιά, η τιμή της τοπικής του μεταβλητής $isleft$ ισούται με 0.

- 3*η επίσκεψη στον κόμβο* t. Ο αλγόριθμος επεξεργάζεται τον κόμβο t, ο οποίος έχει 2 μη επισημασμένα παιδιά.

- 2*η επίσκεψη στον κόμβο* q. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για τον κόμβο u, δηλαδή για το παιδί ενός μη επισημασμένου κόμβου. Κατά συνέπεια, η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου u ισούται με την τιμή της τοπικής μεταβλητής $isleft$ του κόμβου q, δηλαδή με 0.

- 1*η επίσκεψη στον κόμβο* u. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για το αριστερό παιδί του κόμβου u, ο οποίος όμως δεν έχει αριστερό παιδί.

- 2*η επίσκεψη στον κόμβο* u. Ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για το δεξί παιδί του κόμβου u, ο οποίος όμως δεν έχει δεξί παιδί.

- 3*η επίσκεψη στον κόμβο* u. Ο αλγόριθμος επεξεργάζεται τον κόμβο u, ο οποίος αν και είναι ένας εσωτερικός κόμβος που δεν έχει μη επισημασμένα παιδιά, η τιμή της τοπικής του μεταβλητής $isleft$ ισούται με 0.

- 3*η επίσκεψη στον κόμβο* q. Ο αλγόριθμος επεξεργάζεται τον κόμβο q, ο οποίος έχει 1 μη επισημασμένο παιδί.

- 3*η επίσκεψη στον κόμβο* c. Ο αλγόριθμος επεξεργάζεται τον κόμβο c, ο οποίος έχει 1 μη επισημασμένο παιδί.

- 3*η επίσκεψη στον κόμβο* a. Ο αλγόριθμος επεξεργάζεται τον κόμβο a, ο οποίος είναι η ρίζα του δέντρου και δεν έχει αριστερό μη επισημασμένο παιδί. Κατά συνέπεια, Σχήμα 4.1δ΄, ένας ερευνητής εισάγεται στον κόμβο a.

4.3 Ανάλυση αλγορίθμου

4.3.1 Απόδειξη ορθότητας

Λήμμα 4.3.1. *Αν x, y είναι 2 κόμβοι ενός τυχαίου δυαδικού δέντρου T τέτοιοι ώστε, 1. ο x είναι ένας επισημασμένος κόμβος 2. ο x είναι πατέρας του y, τότε η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου y ισούται με 1 αν ο y είναι αριστερό παιδί του κόμβου x ή με 0 αν ο y είναι δεξί παιδί του κόμβου x.*

Απόδειξη. Για κάθε κόμβο y του δέντρου, στην τοπική μεταβλητή $isleft$ καταχωρίζεται η τιμή του 2ου ορίσματος του αλγορίθμου $putSearchers$, όταν αυτός καλείται με 1ο όρισμα τη διεύθυνση του κόμβου y. Έτσι, τη χρονική στιγμή που ο αλγόριθμος επισκέπτεται για 1η φορά τον κόμβο x εκτελείται η 2η γραμμή του αλγορίθμου, όπου η συνθήκη ελέγχου που λαμβάνει χώρα είναι αληθής, οπότε η ροή μεταφέρεται στην αμέσως επόμενη γραμμή (γραμμή 3). Αφού ο κόμβος x είναι, με βάση την υπόθεση του λήμματος ένας επισημασμένος κόμβος, η συνθήκη ελέγχου της 3ης γραμμής του αλγορίθμου είναι αλήθης, οπότε εκτελείται η 4η γραμμή του αλγόριθμου, όπου ο αλγόριθμος καλεί αναδρομικά τον εαυτό του με 1ο όρισμα τη διεύθυνση του αριστερού παιδιού του κόμβου x και 2ο όρισμα την τιμή 1. Κατά συνέπεια, αν ο κόμβος y είναι αριστερό παιδί του κόμβου x, τότε η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου y ισούται με 1. Έπειτα, αφού ο αλγόριθμος επεξεργαστεί κάθε κόμβο που ανήκει στο αριστερό υποδέντρο του κόμβου x, επιστρέφει για 2η φορά σε αυτόν, οπότε εκτελείται η 5η γραμμή του αλγορίθμου, όπου ο αλγόριθμος καλεί αναδρομικά τον εαυτό του με 1ο όρισμα τη διεύθυνση του δεξιού παιδιού του κόμβου x και 2ο όρισμα την τιμή 0. Κατά συνέπεια, αν ο κόμβος y είναι δεξί παιδί του κόμβου x, τότε η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου y ισούται με 0. □

Λήμμα 4.3.2. *Αν x, y είναι 2 κόμβοι ενός τυχαίου δυαδικού δέντρου T τέτοιοι ώστε, 1. ο x είναι ένας μη επισημασμένος κόμβος 2. ο x είναι πατέρας του y, τότε η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου y ισούται με την τιμή της τοπικής μεταβλητής $isleft$ του κόμβου x.*

Απόδειξη. Τη χρονική στιγμή που ο αλγόριθμος επισκέπτεται για 1η φορά τον κόμβο x εκτελείται η 2η γραμμή του αλγορίθμου, όπου η συνθήκη ελέγχου που λαμβάνει χώρα είναι προφανώς αληθής, οπότε η ροή μεταφέρεται στην αμέσως επόμενη γραμμή (γραμμή 3). Αφού ο κόμβος x είναι, με βάση την υπόθεση του λήμματος ένας μη επισημασμένος κόμβος, η συνθήκη ελέγχου της 3ης γραμμής του αλγορίθμου είναι ψευδής, οπότε εκτελείται η 7η γραμμή του αλγόριθμου, όπου ο αλγόριθμος καλεί αναδρομικά τον εαυτό του με 1ο όρισμα τη διεύθυνση του αριστερού παιδιού του κόμβου x και 2ο όρισμα την τιμή της τοπικής μεταβλητής $isleft$ του κόμβου x. Κατά συνέπεια, αν ο κόμβος y είναι αριστερό παιδί του κόμβου x, τότε η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου y ισούται με την τιμή της τοπικής μεταβλητής $isleft$ του κόμβου x. Έπειτα, αφού ο αλγόριθμος επεξεργαστεί κάθε κόμβο που ανήκει στο αριστερό υποδέντρο του κόμβου x, επιστρέφει για 2η φορά σε αυτόν, οπότε εκτελείται η 8η γραμμή του αλγορίθμου, όπου ο αλγόριθμος καλεί αναδρομικά τον εαυτό του με 1ο όρισμα τη διεύθυνση του δεξιού παιδιού του κόμβου x και 2ο όρισμα την τιμή της τοπικής μεταβλητής $isleft$ του κόμβου x. Κατά συνέπεια, αν ο κόμβος y είναι δεξί

παιδί του κόμβου x, τότε η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου y ισούται με την τιμή της τοπικής μεταβλητής $isleft$ του κόμβου x. □

Λήμμα 4.3.3. *Από τα λήμματα 4.3.1 και 4.3.2 προκύπτει ότι η τιμή της τοπικής μεταβλητής* $isleft$ *ισούται με* 1 *για κάθε κόμβο που ανήκει στο αριστερό υποδέντρο του πλησιέστερου επισημασμένου προγόνου του και με* 0 *για κάθε κόμβο που ανήκει στο δεξιό υποδέντρο του πλησιέστερου επισημασμένου προγόνου του.*

Λήμμα 4.3.4. *Ο αλγόριθμος* $putSearchers$ *τοποθετεί από έναν ερευνητή σε κάθε κόμβο του δέντρου ο οποίος ανήκει στο αριστερό υποδέντρο του πλησιέστερου επισημασμένου προγόνου του και ο οποίος δεν έχει μη επισημασμένα παιδιά ή στη ρίζα του δέντρου αν αυτή δεν έχει αριστερό μη επισημασμένο παιδί.*

Απόδειξη. Τη χρονική στιγμή που ο αλγόριθμος επισκέπτεται για 3η φορά έναν κόμβο u,

- αν *ο κόμβος* u *είναι η ρίζα του δέντρου*, τότε η συνθήκη ελέγχου της 10η γραμμής του αλγορίθμου είναι αληθής, οπότε εκτελείται η γραμμή 11, όπου
  - αν *ο κόμβος* u *δεν έχει αριστερό μη επισημασμένο παιδί*, τότε η συνθήκη ελέγχου της 11ης γραμμής του αλγορίθμου είναι αληθής, οπότε ένας ερευνητής τοποθετείται στη ρίζα του δέντρου (γραμμή 12) και αυξάνεται κατά 1 το πλήθος των ερευνητών που απαιτούνται για τον καθαρισμό του δέντρου (γραμμη 13). Έπειτα εκτελείται η 23η γραμμή του αλγορίθμου.
  - *ο κόμβος* u *έχει αριστερό μη επισημασμένο παιδί*, τότε η συνθήκη ελέγχου που λαμβάνει χώρα στη γραμμή 11 είναι ψευδής, οπότε εκτελείται η γραμμή 23.
- αν *ο κόμβος* u *ειναι ένας από τους εσωτερικούς κόμβους του δέντρου*, τότε η συνθήκη ελέγχου της 10ης γραμμής είναι ψευδής, οπότε εκτελειται η γραμμή 16, όπου
  - αν *ο κόμβος* u *δεν έχει μη επισημασμένα παιδιά*, τότε η συνθήκη ελέγχου της 16ης γραμμής του αλγορίθμου είναι αληθής, άρα εκτελείται η 17η γραμμή του αλγορίθμου, όπου
    * αν *ο* u *είναι αριστερό παιδί του πλησιέστερου επισημασμένου προγόνου του*, τότε, με βάση το λήμμα 4.3.3, η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου u ισούται με 1, οπότε η συνθήκη ελέγχου της 17ης γραμμής του αλγορίθμου είναι αληθής. Κατά συνέπεια, ο αλγόριθμος τοποθετεί έναν ερευνητή στον κόμβο u (γραμμή 18) και αυξάνει κατά 1 την τιμή της μεταβλητής που σώζει το πλήθος των ερευνητών που απαιτούνται για τον καθαρισμό του δέντρου (γραμμή 19). Έπειτα, εκτελείται η 23η γραμμή του αλγορίθμου.
    * αν *ο* u *είναι δεξί παιδί του πλησιέστερου επισημασμένου προγόνου του*, τότε, με βάση το λήμμα 4.3.3, η τιμή της τοπικής μεταβλητής $isleft$ του κόμβου u ισούται με 0, οπότε η συνθήκη ελέγχου της 17ης γραμμής του αλγορίθμου είναι ψευδής και εκτελειται η 23η γραμμή του αλγορίθμου.

– αν *ο u έχει τουλάχιστον ένα μη επισημασμένο παιδί*, τότε η συνθήκη ελέγχου της 16ης γραμμής του αλγορίθμου είναι ψευδής, οπότε εκτελείται η 23η γραμμή του αλγορίθμου.

Η γραμμή 23, τέλος, επιστρέφει το πλήθος των ερευνητών που έχουν εισαχθεί μέχρι στιγμής στο δέντρο. □

Λήμμα 4.3.5. *Από το λήμμα 4.3.4 συνεπάγεται ότι, αν k ο αριθμός των επισημασμένων κόμβων ενός τυχαίου δυαδικού δέντρου T, τότε ο αλγόριθμος μας τοποθετεί από έναν ερευνητή σε k διαφορετικούς κόμβους του T.*

4.3.2 Απόδειξη πολυπλοκότητας

Λήμμα 4.3.6. *Η κλήση $putSearchers(u)$, όπου u η ρίζα ενός τυχαίου δυαδικού δέντρου με n κόμβους, απαιτεί χρόνο $\Theta(n)$.*

Απόδειξη. Αρκεί να αποδείξουμε ότι ο χρόνος d που χρειάζεται ο αλγόριθμος μας για να επεξεργαστεί 1 μόνο κόμβο (γραμμές $10-23$) είναι σταθερός και ανεξάρτητος του μεγέθους της εισόδου n. Ο παρακάτω πίνακας συνοψίζει το κόστος εκτελέσεως και το πλήθος εκτελέσεων κάθε γραμμής κατά τη διάρκεια της επεξεργασίας ενός κόμβου u από τον αλγόριθμο $putSearchers$:

Γραμμή	*Κόστος*	*Αριθμός εκτελέσεων*
10	c_{10}	1
11	c_{11}	≤ 1
12	c_{12}	≤ 1
13	c_{13}	≤ 1
16	c_{16}	≤ 1
17	c_{17}	≤ 1
18	c_{18}	≤ 1
19	c_{19}	≤ 1
23	c_{23}	1

$$d \leq c_{10} + max(c_{11} + c_{12} + c_{13}, c_{16} + c_{17} + c_{18} + c_{19}) + c_{23}$$

Αυτός ο χρόνος εκτέλεσης μπορεί να γραφεί στη μορφή a όπου η σταθερά a εξαρτάται από τα διάφορα κόστη c_i των εντολών. Επομένως, είναι μια σταθερή συνάρτηση του n. □

4.4 Αριθμός ερευνητών, κάτω φράγμα

Λήμμα 4.4.1. *Αν k το πλήθος των επισημασμένων κόμβων ενός τυχαίου δυαδικού δέντρου T, το οποίο αποτελείται από n κόμβους, τότε δεν μπορεί να υπάρξει καμία $exclusive\ monotone$ στρατηγική που να καθαρίζει το T με λιγότερους από k ερευνητές.*

Απόδειξη. Ας προσπαθήσουμε να καθαρίσουμε το γράφημα χρησιμοποιώντας έναν ερευνητή, έστω s_1 αυτός. Έστω ότι αυτός φτάνει για πρώτη φορά σε επισημασμένο κόμβο, εκτός της ρίζας, μέσω της ακμής w_1w. Τότε, λόγω της ολίσθησης, η συγκεκριμένη ακμή καθαρίζει, ενώ ο s_1 στέκεται τώρα στον επισημασμένο κόμβο w. Όμως στον w προσπίπτουν πλέον 2 μολυσμένες ακμές, έστω ww_2 και ww_3 αυτές, με αποτέλεσμα ο s_1 να μη μπορεί πλέον να απομακρυνθεί από τον w, αφού αν το κάνει θα υπάρξει επαναμόλυνση. Έστω ένας δεύτερος ερευνητής, s_2, ο οποίος στέκεται σε κάποιον από τους κόμβους w_2 ή w_3, π.χ. στον w_2. Τότε, εκτός της w_1w που θα έχει καθαρίσει λόγω της ολίσθησης, καθαρή θα είναι και η ww_2 λόγω της ταυτόχρονης ύπαρξης ενός ερευνητή και στα δυο της άκρα. Πλέον, στον w προσπίπτει μία μολυσμένη ακμή μέσω της οποίας ο s_1 μπορεί να απομακρυνθεί από τον w χωρίς να υπάρξει κίνδυνος επαναμόλυνσης. Παρατηρείστε κάτι: Αφού το T είναι άκυκλο, οι s_1, s_2 βρίσκονται σε γειτονικοούς κόμβους του w, ενώ δύο ερευνητές δε μπορουν να βρεθούν στον ίδιο κόμβο ταυτόχρονα, αυτό σημαίνει ότι οι δύο αυτοί ερευνητές θα εισέρχονται για πρώτη φορά σε έναν νέο κόμβο από την ίδια ακμή.

Έστω, τώρα u_1u, η ακμή μέσω της οποίας ένας από τους ερευνητές, π.χ. ο s_1 εισέρχεται για πρώτη φορά σε έναν άλλον, εκτός της ρίζας, επισημασμένο κόμβο u. Η ακμή αυτή, λόγω της ολίσθησης καθαρίζει, ενώ στον u προσπίπτουν πλέον 2 μολυσμένες ακμές, οι uu_2 και uu_3. Για τον ίδιο λόγο με πριν, ένας ερευνητής θα πρέπει να βρίσκεται σε έναν από τους κόμβους u_2, u_3. Όμως, αυτός ο ερευνητής δε μπορεί να είναι ο s_2, καθώς εισέρχεται για πρώτη φορά στον u μέσω της u_1u. Έστω, λοιπόν, ένας τρίτος ερευνητής s_3 ο οποίος στέκεται στον κόμβο u_2. Τότε, ο s_2 μπορεί να απομακρυνθεί από τον u χωρίς να υπάρξει επαναμόλυνση. Όμοια με πριν, οι s_2 και s_3 εισέρχονται για πρώτη φορά σε έναν νέο κόμβο από την ίδια ακμή. Αφού το ίδιο ισχύει και για τους s_1, s$_2$, το ίδιο θα ισχύει και για τους s_1και s_3. Συνοψίζοντας, μέχρι στιγμής, έχουμε επισκεφθεί δύο επισημασμένους κόμβους χρησιμοποιώντας τρεις ερευνητές.

Συνεπάγεται λοιπόν ότι, εκτός του αρχικού ερευνητή, χρειάζεται από ένας επιπλέον ερευνητής για κάθε επισημασμένο κόμβο, εκτός της ρίζας. Κατά συνέπεια, απαιτούνται τουλάχιστον $(k-1)+1 = k$ ερευνητές. Η ρίζα είναι ένας κόμβος στον οποίο προσπίπτουν αρχικά 2 μολυσμένες ακμές. Άρα, θα μπορούσε κάποια στιγμή κάποιος από τους ήδη υπάρχοντες ερευνητές να προσεγγίσει τη ρίζα v μέσω της ακμής vv_1, η οποία λόγω της ολίσθησης καθαρίζει. Πλέον, στη ρίζα του δέντρου προσπίπτει μία μολυσμένη ακμή, η vv_2 κατά μήκος της οποίας θα μπορούσε να ολισθήσει ο ερευνητής για να την καθαρίσει χωρίς να υπάρχει κίνδυνος επαναμόλυνσης. □

Κεφάλαιο 5

Καθαρισμός, φάση 1

Η διαδικασία περιγράφεται από τον αλγόριθμο $postorderCleaning$. Η αρχική κλήση είναι $postorderCleaning(root)$, ενώ πριν αυτό συμβεί θα πρέπει να έχει κληθεί ο αλγόριθμος $putSearchers$.

5.1 Ο αλγόριθμος postorderCleaning

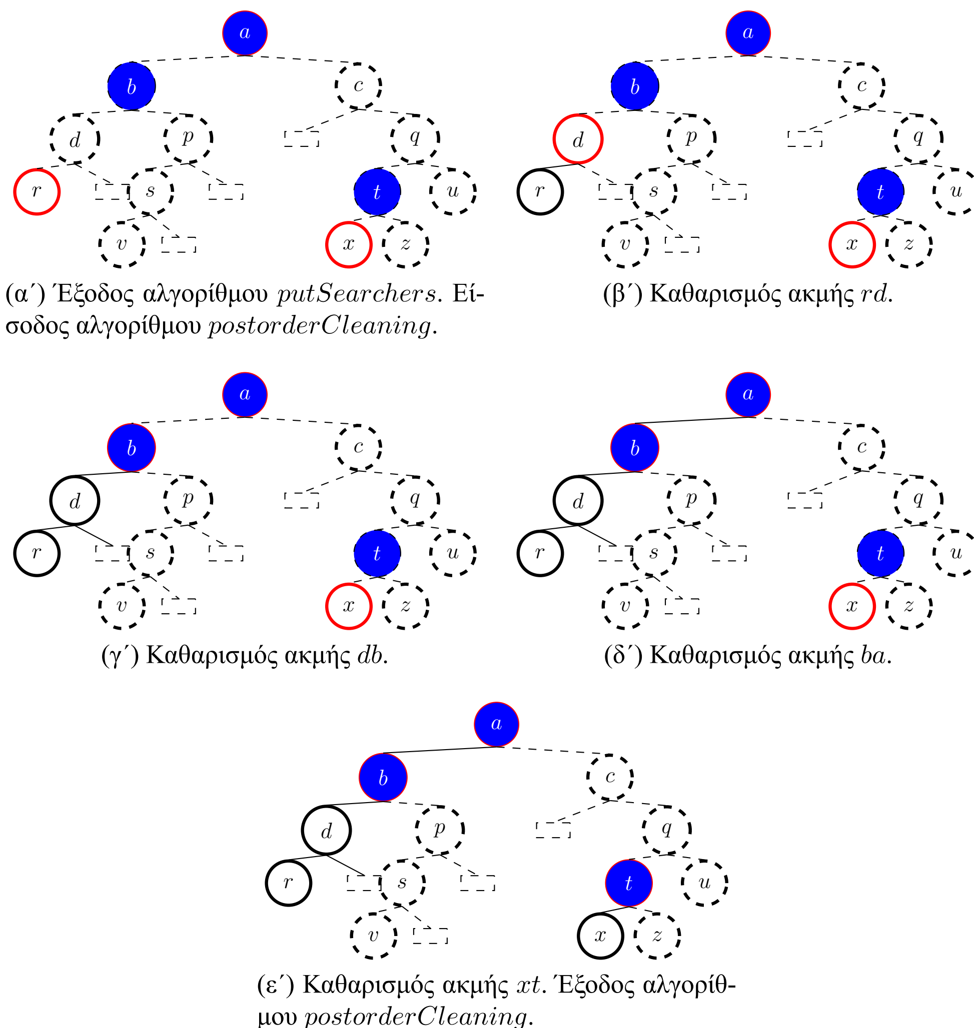
Είσοδος: ο κόμβος u του οποίου το υποδέντρο T_u θα διαπεράσουμε
Έξοδος:

1: **function** $postorderCleaning(u)$
2:     **if** ο u υπάρχει **then**
3:         $postorderCleaning$(αριστερό παιδί του u)
4:         $postorderCleaning$(δεξί παιδί του u)
5:         **if** ο u είναι ένας μη επισημασμένος κόμβος με ερευνητή **then**
6:             ολίσθηση του ερευνητή κατά μήκος της ακμής uv, όπου v ο πατέρας του u
7:             καθαρισμός ακμής uv
8:             **if** υπάρχει ερευνητής σε κάποιον γειτονικό κόμβο w του v **then**
9:                 καθάρισε την ακμή vw
10:             **end if**
11:         **end if**
12:     **end if**
13: **end function**

Algorithm 3: Καθαρισμός, φάση 1.

5.2 Παράδειγμα εκτέλεσης αλγορίθμου

Για το τυχαίο δυαδικό δέντρο του Σχήματος 5.1α΄, εφαρμόζουμε τον αλγόριθμο $postorderCleaning$:



(α΄) Έξοδος αλγορίθμου $putSearchers$. Είσοδος αλγορίθμου $postorderCleaning$.

(β΄) Καθαρισμός ακμής rd.

(γ΄) Καθαρισμός ακμής db.

(δ΄) Καθαρισμός ακμής ba.

(ε΄) Καθαρισμός ακμής xt. Έξοδος αλγορίθμου $postorderCleaning$.

Σχήμα 5.1: Στιγμιότυπο $postorderCeaning$.

- *Επίσκεψη στον κόμβο* r (Σχήμα 5.1α΄). Αφού ο r είναι ένας μη επισημασμένος κόμβος, ο ερευνητής που βρίσκεται σε αυτόν, ολισθαίνει κατά μήκος της μοναδικής μολυσμένης ακμής που προσπίπτει στον r. Είναι η ακμή rd, η οποία συνδέει τον κόμβο r με τον πατέρα του. Κατά συνέπεια, Σχήμα 5.1β΄, η ακμή rd καθαρίζει και ο ερευνητής εισέρχεται στον κόμβο d.

- *Επίσκεψη στον κόμβο* d (Σχήμα 5.1β΄). Αφού ο d είναι ένας μη επισημασμένος κόμβος, ο ερευνητής που βρίσκεται σε αυτόν, ολισθαίνει κατά μήκος της μοναδικής μολυσμένης ακμής που προσπίπτει στον d. Είναι η ακμή db, η οποία συνδέει τον κόμβο d με τον πατέρα του. Κατά συνέπεια, Σχήμα 5.1γ΄, η ακμή db καθαρίζει και ο ερευνητής εισέρχεται στον κόμβο b. Η άφιξη του ερευνητή στον κόμβο b συνεπάγεται την ύπαρξη ενός ερευνητή σε κάθε άκρο της μολυσμένης ακμής ba. Κατά συνέπεια, Σχήμα 5.1δ΄,η ακμή ba καθαρίζει.

- *Επίσκεψη στον κόμβο v* (Σχήμα 5.1δ΄), ο οποίος δεν έχει ερευνητή.
- *Επίσκεψη στον κόμβο s* (Σχήμα 5.1δ΄), ο οποίος δεν έχει ερευνητή.
- *Επίσκεψη στον κόμβο p* (Σχήμα 5.1δ΄), ο οποίος δεν έχει ερευνητή.
- *Επίσκεψη στον κόμβο b* (Σχήμα 5.1δ΄), ο οποίος είναι ένας επισημασμένος κόμβος.
- *Επίσκεψη στον κόμβο x* (Σχήμα 5.1δ΄), ο οποίος είναι ένας μη επισημασμένος κόμβος. Έτσι, ο ερευνητής που βρίσκεται σε αυτόν, ολισθαίνει κατά μήκος της μοναδικής μολυσμένης ακμής που προσπίπτει στον x. Είναι η ακμή xt, η οποία συνδέει τον κόμβο x με τον πατέρα του. Κατά συνέπεια, Σχήμα 5.1ε΄, η ακμή xt καθαρίζει και ο ερευνητής εισέρχεται στον κόμβο t.
- *Επίσκεψη στον κόμβο z* (Σχήμα 5.1ε΄), ο οποίος δεν έχει ερευνητή.
- *Επίσκεψη στον κόμβο t* (Σχήμα 5.1ε΄), ο οποίος είναι ένας επισημασμένος κόμβος.
- *Επίσκεψη στον κόμβο u* (Σχήμα 5.1ε΄), ο οποίος δεν έχει ερευνητή.
- *Επίσκεψη στον κόμβο q* (Σχήμα 5.1ε΄), ο οποίος δεν έχει ερευνητή.
- *Επίσκεψη στον κόμβο c* (Σχήμα 5.1ε΄), ο οποίος δεν έχει ερευνητή.
- *Επίσκεψη στον κόμβο a* (Σχήμα 5.1ε΄), ο οποίος είναι ένας επισημασμένος κόμβος.

5.3 Ανάλυση αλγορίθμου

5.3.1 Απόδειξη ορθότητας

Λήμμα 5.3.1. *Αν $u_1, u_2, \ldots, u_n$ μια ακολουθία από μη επισημασμένους κόμβους ενός τυχαίου δυαδικού δέντρου T τέτοιοι ώστε, 1. ο u_i είναι πατέρας του u_{i-1}, για κάθε $2 \leq i \leq n$ 2. η ακμή $u_{-1}u_i$ είναι μολυσμένη, για κάθε $2 \leq i \leq n$ 3. μόνο στον κόμβο u_1 υπάρχει ερευνητής 4. στον u_1 ενδέχεται να προσπίπτουν άλλες μία ή δύο ακόμη καθαρές ακμές 5. στον u_n προσπίπτει ακόμα μία μολυσμένη ακμή, τότε η επεξεργασία του κόμβου u_i, για κάθε $1 \leq i \leq n-1$, συνεπάγεται την ολίσθηση του ερευνητή του κόμβου u_i κατά μήκος της μολυσμένης ακμής u_iu_{i+1} και κατά συνέπεια τον καθαρισμό αυτής.*

Απόδειξη. Θα αποδείξουμε το λήμμα χρησιμοποιώντας τη μέθοδο της μαθηματικής επαγωγής.

- *Βάση της επαγωγής.* Το λήμμα ισχύει μετά την επεξεργασία του κόμβου u_1. Αυτός, με βάση την υπόθεση του λήμματος είναι ένας μη επισημασμένος κόμβος ο οποίος έχει ερευνητή. Κατά συνέπεια, κατά την επεξεργασία του κόμβου u_1 η συνθήκη ελέγχου της 5ης γραμμής του αλγορίθμου είναι αληθής, οπότε ο ερευνητής απομακρύνεται από τον κόμβο u_1 ολισθαίνοντας κατά μήκος της μολυσμένης ακμής u_1u_2 (γραμμή 6), με αποτέλεσμα τον καθαρισμό αυτής (γραμμή 7).

- *Υποθέτουμε επαγωγικά* ότι το λήμμα ισχύει μετά από την επεξεργασία του κόμβου u_k, $k \leq n-2$, υποθέτουμε δηλαδή ότι μετά την επεξεργασία του κόμβου u_k, ισχύει ότι ο ερευνητής του κόμβου u_k ολισθαίνει κατά μήκος της μολυσμένης ακμής $u_k u_{k+1}$ καθαρίζοντάς την.

- Θα αποδείξουμε ότι το λήμμα συνεχίζει να ισχύει και μετά την επεξεργασία του κόμβου u_{k+1}, θα αποδείξουμε δηλαδή ότι μετά την επεξεργασία του κόμβου u_{k+1} ο ερευνητής που βρισκόταν στον κόμβο u_{k+1} βρίσκεται πλέον στον κόμβο u_{k+2}, και η ακμή $u_{k+1}u_{k+2}$ είναι καθαρή. Από την υπόθεση του λήμματος έχουμε ότι ο κόμβος u_{k+1} είναι μη επισημασμένος, ενώ από την επαγωγική υπόθεση ισχύει ότι στον κόμβο u_{k+1} υπάρχει ερευνητής. Κατά συνέπεια, κατά την επεξεργασία του κόμβου u_{k+1} η συνθήκη ελέγχου της 5ης γραμμής του αλγορίθμου είναι αληθής, οπότε ο ερευνητής απομακρύνεται από τον κόμβο u_{k+1} ολισθαίνοντας κατά μήκος της μολυσμένης ακμής $u_{k+1}u_{k+2}$ (γραμμή 6), με αποτέλεσμα τον καθαρισμό αυτής (γραμμή 7). □

Λήμμα 5.3.2. *Ο αλγόριθμος $postorderCleaning$ κινεί κάθε ερευνητή προς τον πλησιέστερο επισημασμένου πρόγονό του, ενώ καθαρίζει κάθε ακμή που ανήκει στο αριστερό υποδέντρο του πλησιέστερου επισημασμένου προγόνου της.*

Απόδειξη. Θα αποδείξουμε το λήμμα χρησιμοποιώντας τη μέθοδο της μαθηματικής επαγωγής.

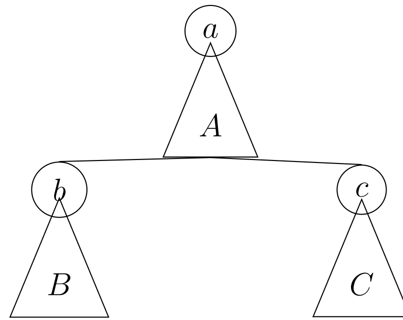
- *Βάση της επαγωγής.* Για δυαδικό δέντρο με 1 μόνο επισημασμένο κόμβο το λήμμα ισχύει.

- *Υποθέτουμε επαγωγικά* ότι το λήμμα ισχύει για κάθε δυαδικό δέντρο με k επισημασμένους κόμβους.

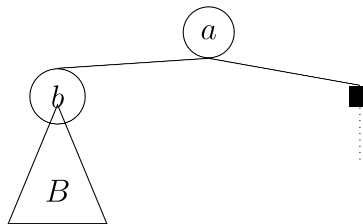
- Θα αποδείξουμε ότι το λήμμα ισχύει και για κάθε δυαδικό δέντρο με $k+1$ επισημασμένους κόμβους. Έστω δυαδικό δέντρο T αποτελούμενο από n κόμβους, όπου A, B και C υποδέντρα του T τέτοια ώστε οι επισημασμένοι κόμβοι a, b και c είναι ρίζες των υποδέντρων A, B και C αντίστοιχα, και $1, x \leq k, y \leq k$ είναι το πλήθος των επισημασμένων κόμβων των υποδέντρων A, B και C αντίστοιχα (Σχήμα 5.2).

  1. Ξεκινάμε με το αριστερό υποδέντρο του κόμβου a και διακρίνουμε τις ακόλουθες περιπτώσεις:

     – *Έστω a ο πατέρας του b (Σχήμα 5.3).* Τότε, με βάση την επαγωγική μας υπόθεση, το λήμμα ισχύει για το υποδέντρο B, αφού αυτό αποτελείται από το πολύ k επισημασμένους κόμβους. Έτσι, μετά την επεξεργασία της ρίζας του υποδέντρου B, δηλαδή του κόμβου b, από ένας ερευνητής υπάρχει σε κάθε επισημασμένο κόμβο του B, ενώ κάθε ακμή του υποδέντρου B που ανήκει στο αριστερό υποδέντρο του πλησιέστερου επισημασμένου προγόνου της είναι καθαρή. Από την άλλη, αφού το αριστερό παιδί του κόμβου a είτε δεν υπάρχει είτε είναι ένας επισημασμένος κόμβος, με βάση το λήμμα 3.3.1, ένας ερευνητής θα βρίσκεται από την αρχή στον κόμβο a. Έτσι, κατά



Σχήμα 5.2

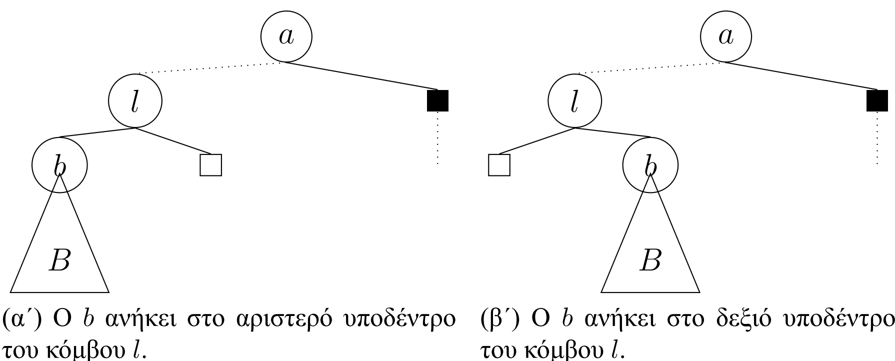


Σχήμα 5.3: Ο a είναι πατέρας του b.

την επεξεργασία του αριστερού παιδιού του κόμβου b, η απομάκρυνση του ερευνητή από αυτό και η άφιξή του στον κόμβο b συνεπάγεται την ύπαρξη ενός ερευνητή σε κάθε άκρο της ακμής ba. Κατά συνέπεια, η συνθήκη ελέγχου της 8ης γραμμής του αλγορίθμου είναι αληθής, οπότε η ακμή ba καθαρίζει (γραμμή 9).

– *Έστω l το αριστερότερο φύλλο του υποδέντρου A.* Αφού ο κόμβος l δεν έχει μη επισημασμένα παιδιά και ανήκει στο αριστερό υποδέντρο του πλησιέστερου επισημασμένου προγόνου του (κόμβος a), με βάση το λήμμα 3.3.1, υπάρχει σε αυτόν ένας ερευνητής.

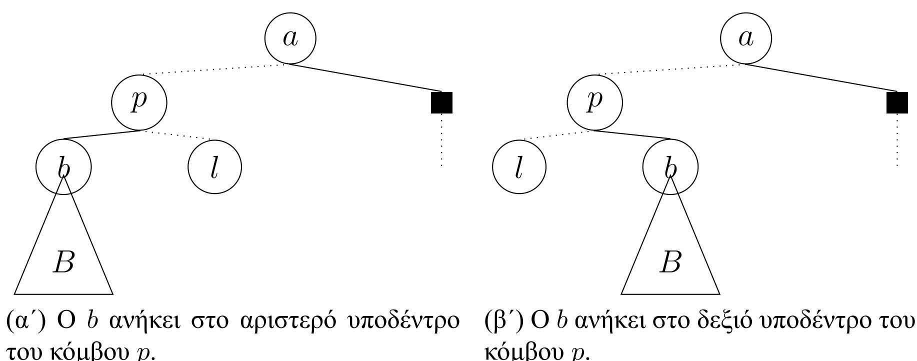
  * *Ο l είναι πατέρας του κόμβου b (Σχήμα 5.4).* Τότε, με βάση την επαγωγική μας υπόθεση, το λήμμα ισχύει για το υποδέντρο B, αφού αυτό αποτελείται από το πολύ k επισημασμένους κόμβους. Έτσι, μετά την επεξεργασία της ρίζας του υποδέντρου B, δηλαδή του κόμβου b, από ένας ερευνητής υπάρχει σε κάθε επισημασμένο κόμβο του B, ενώ κάθε ακμή του υποδέντρου B που ανήκει στο αριστερό υποδέντρο του πλησιέστερου επισημασμένου προγόνου της είναι καθαρή. Επιπλέον, κατά την επεξεργασία του αριστερού παιδιού του κόμβου p, η απομάκρυνση του ερευνητή από αυτό και η άφιξή του στον κόμβο b συνεπάγεται την ύπαρξη ενός ερευνητή σε κάθε άκρο της ακμής bl. Κατά συνέπεια, η συνθήκη ελέγχου της 8ης γραμμής του αλγορίθμου είναι αληθής, οπότε η ακμή bl καθαρίζει (γραμμή 9). Τέλος, με βάση το λήμμα 5.3.1, μετά την επεξεργασία του αριστερού παιδιού του κόμβου a, ο ερευνητής του κόμβου l βρίσκεται πλέον στον κόμβο a, ενώ κάθε ακμή που βρίσκεται μεταξύ των κόμβων l και a ειναι πλέον καθαρή.



(α΄) Ο b ανήκει στο αριστερό υποδέντρο του κόμβου l.

(β΄) Ο b ανήκει στο δεξιό υποδέντρο του κόμβου l.

Σχήμα 5.4: Ο l είναι πατέρας του b.

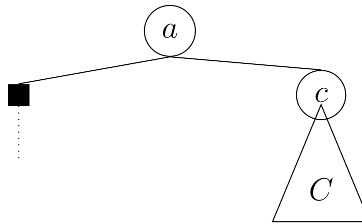


(α΄) Ο b ανήκει στο αριστερό υποδέντρο του κόμβου p.

(β΄) Ο b ανήκει στο δεξιό υποδέντρο του κόμβου p.

Σχήμα 5.5: Ο l δεν είναι πατέρας του b. Έστω p ο πατέρας του b.

* *Ο l δεν είναι πατέρας του κόμβου b (Σχήμα 5.5).* Τότε, έστω p ο πατέρας του κόμβου b.
  · *Ο b ανήκει στο αριστερό υποδέντρο του κόμβου p (Σχήμα 5.5α΄)* Τότε, με βάση την επαγωγική μας υπόθεση, το λήμμα ισχύει για το υποδέντρο B, αφού αυτό αποτελείται από το πολύ k επισημασμένους κόμβους. Έτσι, μετά την επεξεργασία της ρίζας του υποδέντρου B, δηλαδή του κόμβου b, από ένας ερευνητής υπάρχει σε κάθε επισημασμένο κόμβο του B, ενώ κάθε ακμή του υποδέντρου B που ανήκει στο αριστερό υποδέντρο του πλησιέστερου επισημασμένου προγόνου της είναι καθαρή. Έπειτα, και με βάση το λήμμα 5.3.1, μετά την επεξεργασία του δεξιού παιδιού του κόμβου p, ο ερευνητής του κόμβου l βρίσκεται πλέον στον κόμβο p, ενώ κάθε ακμή που βρίσκεται μεταξύ των κόμβων l και p ειναι πλέον καθαρή. Επιπλέον, κατά την επεξεργασία του δεξιού παιδιού του κόμβου p, η απομάκρυνση του ερευνητή από αυτό και η άφιξή του στον κόμβο p συνεπάγεται την ύπαρξη ενός ερευνητή σε κάθε άκρο της ακμής pb. Κατά συνέπεια, η συνθήκη ελέγχου της 8ης γραμμής του αλγορίθμου είναι αληθής, οπότε η ακμή pb καθαρίζει (γραμμή 9). Τέλος, με βάση και πάλι το



Σχήμα 5.6: Ο a είναι ο πατέρας του c.

λήμμα 5.3.1, μετά την επεξεργασία του αριστερού παιδιού του κόμβου a, ο ερευνητής του κόμβου p βρίσκεται πλέον στον κόμβο a, ενώ κάθε ακμή που βρίσκεται μεταξύ των κόμβων l και a ειναι πλέον καθαρή.

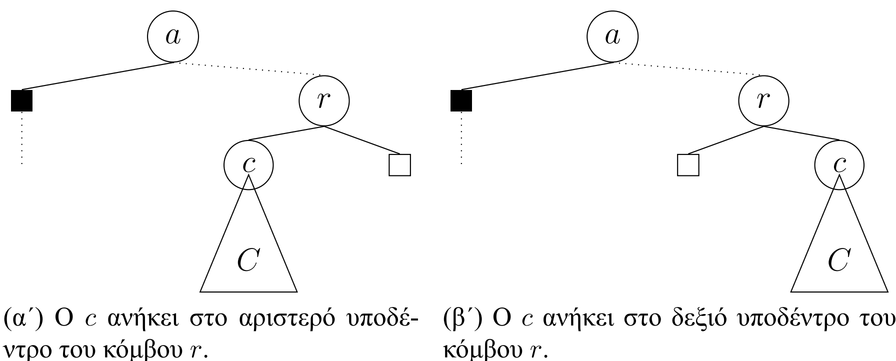
· *Ο b ανήκει στο δεξιό υποδέντρο του κόμβου p (Σχήμα 5.5β΄).* Η απόδειξη είναι εντελώς συμμετρική.

- Συνεχίζουμε με το δεξιό υποδέντρο του κόμβου a, όπου διακρίνουμε τις ακόλουθες περιπτώσεις:

  – *Έστω a ο πατέρας του c (Σχήμα 5.6).* Τότε, με βάση την επαγωγική μας υπόθεση, το λήμμα ισχύει για το υποδέντρο C, αφού αυτό αποτελείται από το πολύ k επισημασμένους κόμβους. Έτσι, μετά την επεξεργασία της ρίζας του υποδέντρου C, δηλαδή του κόμβου c, από ένας ερευνητής υπάρχει σε κάθε επισημασμένο κόμβο του C, ενώ κάθε ακμή του υποδέντρου C που ανήκει στο αριστερό υποδέντρο του πλησιέστερου επισημασμένου προγόνου της είναι καθαρή. Επιπλέον, κατά την επεξεργασία του αριστερού παιδιού του κόμβου c, η απομάκρυνση του ερευνητή από αυτό και η άφιξή του στον κόμβο c συνεπάγεται την ύπαρξη ενός ερευνητή σε κάθε άκρο της ακμής ca. Κατά συνέπεια, η συνθήκη ελέγχου της 8ης γραμμής του αλγορίθμου είναι αληθής, οπότε η ακμή ca καθαρίζει (γραμμή 9). Τέλος, κατά την επεξεργασία του κόμβου a η συνθήκη ελέγχου της 5ης γραμμής του αλγορίθμου είναι ψευδής, αφού αυτός ως ρίζα του δέντρου είναι ένας επισημασμένος κόμβος.

  – *Έστω r το δεξιότερο φύλλο του υποδέντρου A.*

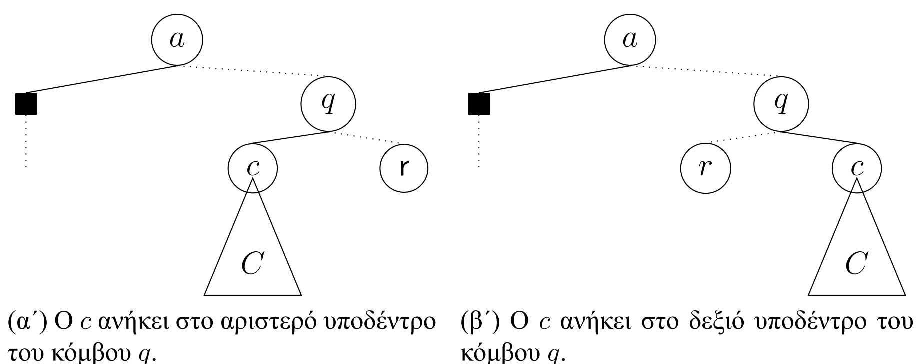
    * *Ο r είναι πατέρας του κόμβου c (Σχήμα 5.7).* Τότε, με βάση την επαγωγική μας υπόθεση, το λήμμα ισχύει για το υποδέντρο C, αφού αυτό αποτελείται από το πολύ k επισημασμένους κόμβους. Έτσι, μετά την επεξεργασία της ρίζας του υποδέντρου C, δηλαδή του κόμβου c, από ένας ερευνητής υπάρχει σε κάθε επισημασμένο κόμβο του C, ενώ κάθε ακμή του υποδέντρου C που ανήκει στο αριστερό υποδέντρο του πλησιέστερου επισημασμένου προγόνου της είναι καθαρή. Έπειτα, κατά την επεξεργασία του κόμβου r και όλων των προγόνων του, εκτός του κόμβου a, η συνθήκη ελέγχου της 5ης γραμμής του αλγορίθμου είναι ψευδής, αφού κανένας από αυτούς τους κόμβους δεν έχει ερευνητή. Τέλος, κατά την επεξεργασία του κόμβου a, η



(α′) Ο c ανήκει στο αριστερό υποδέντρο του κόμβου r.

(β′) Ο c ανήκει στο δεξιό υποδέντρο του κόμβου r.

Σχήμα 5.7: Ο r είναι πατέρας του c.



(α′) Ο c ανήκει στο αριστερό υποδέντρο του κόμβου q.

(β′) Ο c ανήκει στο δεξιό υποδέντρο του κόμβου q.

Σχήμα 5.8: Ο r δεν είναι πατέρας του c. Έστω q ο πατέρας του κόμβου c.

συνθήκη ελέγχου της 5ης γραμμής του αλγορίθμου είναι ψευδής, αφού ο κόμβος a ως ρίζα του δέντρου είναι ένας επισημασμένος κόμβος.

* *Ο r δεν είναι πατέρας του κόμβου c. (Σχήμα 5.8).* Τότε, έστω q ο πατέρας του κόμβου c.

  · *Ο c ανήκει στο αριστερό υποδέντρο του κόμβου q (Σχήμα 5.8α′).* Τότε, με βάση την επαγωγική μας υπόθεση, το λήμμα ισχύει για το υποδέντρο C, αφού αυτό αποτελείται από το πολύ k επισημασμένους κόμβους. Έτσι, μετά την επεξεργασία της ρίζας του υποδέντρου C, δηλαδή του κόμβου c, από ένας ερευνητής υπάρχει σε κάθε επισημασμένο κόμβο του C, ενώ κάθε ακμή του υποδέντρου C που ανήκει στο αριστερό υποδέντρο του πλησιέστερου επισημασμένου προγόνου της είναι καθαρή. Έπειτα, κατά την επεξεργασία του κόμβου r και όλων των προγόνων του, εκτός του κόμβου a, η συνθήκη ελέγχου της 5ης γραμμής του αλγορίθμου είναι ψευδής, αφού κανένας από αυτούς τους κόμβους δεν έχει ερευνητή. Τέλος, κατά την επεξεργασία του κόμβου a, η συνθήκη ελέγχου της 5ης γραμμής του αλγορίθμου είναι ψευδής, διότι ο a ως ρίζα του δέντρου είναι ένας επισημασμένος κόμβος.

· *Ο c ανήκει στο δεξιό υποδέντρο του κόμβου q (Σχήμα 5.8β΄).* Η απόδειξη είναι εντελώς συμμετρική. □

5.3.2 Απόδειξη πολυπλοκότητας

Λήμμα 5.3.3. *Η κλήση $postorderCleaning(u)$, όπου u η ρίζα ενός τυχαίου δυαδικού δέντρου T με n κόμβους, απαιτεί χρόνο $\Theta(n)$.*

Απόδειξη. Αρκεί να αποδείξουμε ότι ο χρόνος d που χρειάζεται ο αλγόριθμος μας για να επεξεργαστεί 1 μόνο κόμβο (γραμμές $5-9$) είναι σταθερός και ανεξάρτητος του μεγέθους της εισόδου n. Ο παρακάτω πίνακας συνοψίζει το κόστος εκτελέσεως και το πλήθος εκτελέσεων κάθε γραμμής κατά τη διάρκεια της επεξεργασίας ενός κόμβου u από τον αλγόριθμο $postorderCleaning$:

Γραμμή	*Κόστος*	*Αριθμός εκτελέσεων*
5	c_5	1
6	c_6	≤ 1
7	c_7	≤ 1
8	c_8	≤ 1
9	c_9	1

$$d \leq c_5 + c_6 + c_7 + c_8 + c_9$$

Αυτός ο χρόνος εκτέλεσης μπορεί να γραφεί στη μορφή a όπου η σταθερά a εξαρτάται από τα διάφορα κόστη c_i των εντολών. Επομένως, είναι μια σταθερή συνάρτηση του n. □

Κεφάλαιο 6

Καθαρισμός, φάση 2

Η διαδικασία περιγράφεται από τον αλγόριθμο $preorderCleaning$. Η αρχική κλήση είναι $preorderCleaning(root)$, ενώ πριν αυτό συμβεί θα πρέπει να έχει κληθεί ο αλγόριθμος $postorderCleaning$.

6.1 Ο αλγόριθμος preorderCleaning

Είσοδος: ο κόμβος u του οποίου το υποδέντρο T_u θα διαπεράσουμε
Έξοδος:

1: **function** $preorderCleaning(u)$
2:     **if** ο u υπάρχει **then**
3:         **if** uv η μοναδική μολυσμένη ακμή που προσπίπτει στον κόμβο u **then**
4:             ολίσθηση του ερευνητή του κόμβου u κατά μήκος της ακμής uv
5:             καθάρισε την ακμή uv
6:             **if** υπάρχει ερευνητής σε κάποιον γειτονικό κόμβο w του v **then**
7:                 καθάρισε την ακμή vw
8:             **end if**
9:             $preorderCleaning$(αριστερό παιδί του u)
10:             $preorderCleaning$(δεξί παιδί του u)
11:         **end if**
12:     **end if**
13: **end function**

Algorithm 4: Καθαρισμός, φάση 2.

6.2 Παράδειγμα εκτέλεσης αλγορίθμου

Για το τυχαίο δυαδικό δέντρο του Σχήματος 6.1α΄, εφαρμόζουμε τον αλγόριθμο $preorderCleaning$:

- *Επίσκεψη στον κόμβο* a (Σχήμα 6.1α΄). Ο ερευνητής του κόμβου ολισθαίνει κατά μήκος της μοναδικής μολυσμένης ακμής που προσπίπτει στον a. Είναι η ακμή ac, η οποία συνδέει τον κόμβο a με το δεξί του παιδί. Κατά συνέπεια, Σχήμα 6.1β΄, η ακμή ac καθαρίζει και ο ερευνητής εισέρχεται στον κόμβο c.

- *Επίσκεψη στον κόμβο* b (Σχήμα 6.1β΄). Ο ερευνητής του κόμβου ολισθαίνει κατά μήκος της μοναδικής μολυσμένης ακμής που προσπίπτει στον b. Είναι η ακμή bp, η οποία συνδέει τον κόμβο b με το δεξί του παιδί. Κατά συνέπεια, Σχήμα 6.1γ΄, η ακμή bp καθαρίζει και ο ερευνητής εισέρχεται στον κόμβο p.

- *Επίσκεψη στον κόμβο* d (Σχήμα 6.1γ΄). Όλες οι ακμές που προσπίπτουν στον κόμβο d είναι καθαρές.

- *Επίσκεψη στον κόμβο* r (Σχήμα 6.1γ΄). Κάθε ακμή που προσπίπτει στον κόμβο r είναι καθαρή.

- *Επίσκεψη στον κόμβο* p (Σχήμα 6.1γ΄). Ο ερευνητής του κόμβου ολισθαίνει κατά μήκος της μοναδικής μολυσμένης ακμής που προσπίπτει στον p. Είναι η ακμή ps, η οποία συνδέει τον κόμβο p με το αριστερό του παιδί. Κατά συνέπεια, Σχήμα 6.1δ΄, η ακμή ps καθαρίζει και ο ερευνητής εισέρχεται στον κόμβο s.

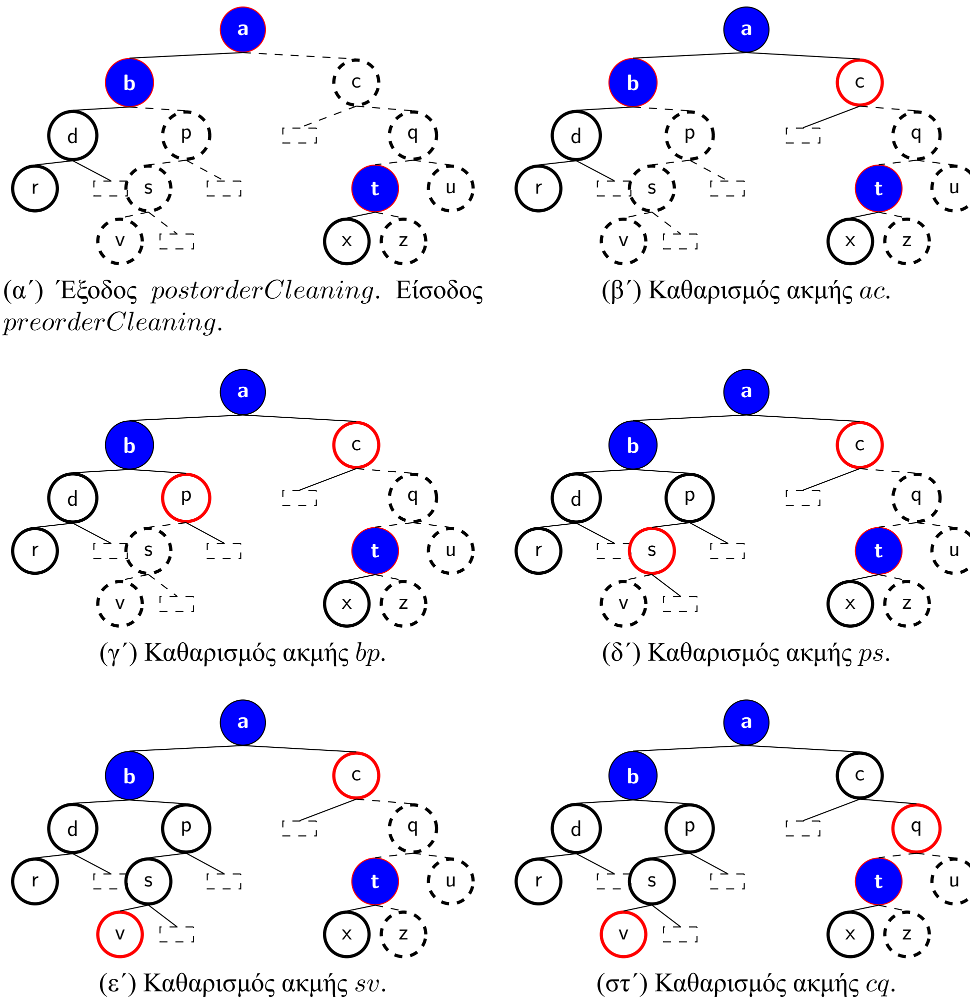
- *Επίσκεψη στον κόμβο* s (Σχήμα 6.1δ΄). Ο ερευνητής του κόμβου ολισθαίνει κατά μήκος της μοναδικής μολυσμένης ακμής που προσπίπτει στον s. Είναι η ακμή sv, η οποία συνδέει τον κόμβο s με το αριστερό του παιδί. Κατά συνέπεια, Σχήμα 6.1ε΄, η ακμή sv καθαρίζει και ο ερευνητής εισέρχεται στον κόμβο v.

- *Επίσκεψη στον κόμβο* v (Σχήμα 6.1ε΄). Καμία μολυσμένη ακμή δεν προσπίπτει στον κόμβο v.

- *Επίσκεψη στον κόμβο* c (Σχήμα 6.1ε΄). Ο ερευνητής του κόμβου ολισθαίνει κατά μήκος της μοναδικής μολυσμένης ακμής που προσπίπτει στον c. Είναι η ακμή cq, η οποία συνδέει τον κόμβο c με το δεξί του παιδί. Κατά συνέπεια, Σχήμα 6.1στ΄, η ακμή cq καθαρίζει και ο ερευνητής εισέρχεται στον κόμβο q. Η άφιξη του ερευνητή στον κόμβο q συνεπάγεται την ύπαρξη ενός ερευνητή σε κάθε άκρο της μολυσμένης ακμής qt. Κατά συνέπεια, Σχήμα 6.1ζ΄,η ακμή qt καθαρίζει.

- *Επίσκεψη στον κόμβο* q (Σχήμα 6.1ζ΄). Ο ερευνητής του κόμβου ολισθαίνει κατά μήκος της μοναδικής μολυσμένης ακμής που προσπίπτει στον q. Είναι η ακμή qu, η οποία συνδέει τον κόμβο q με το δεξί του παιδί. Κατά συνέπεια, Σχήμα 6.1η΄, η ακμή qu καθαρίζει και ο ερευνητής εισέρχεται στον κόμβο u.

- *Επίσκεψη στον κόμβο* t (Σχήμα 6.1η΄). Ο ερευνητής του κόμβου ολισθαίνει κατά μήκος της μοναδικής μολυσμένης ακμής που προσπίπτει στον t. Είναι η ακμή tz, η οποία συνδέει τον κόμβο t με το δεξί του παιδί. Κατά συνέπεια, Σχήμα 6.1θ΄, η ακμή tz καθαρίζει και ο ερευνητής εισέρχεται στον κόμβο z.



(α΄) Έξοδος $postorderCleaning$. Είσοδος $preorderCleaning$.

(β΄) Καθαρισμός ακμής ac.

(γ΄) Καθαρισμός ακμής bp.

(δ΄) Καθαρισμός ακμής ps.

(ε΄) Καθαρισμός ακμής sv.

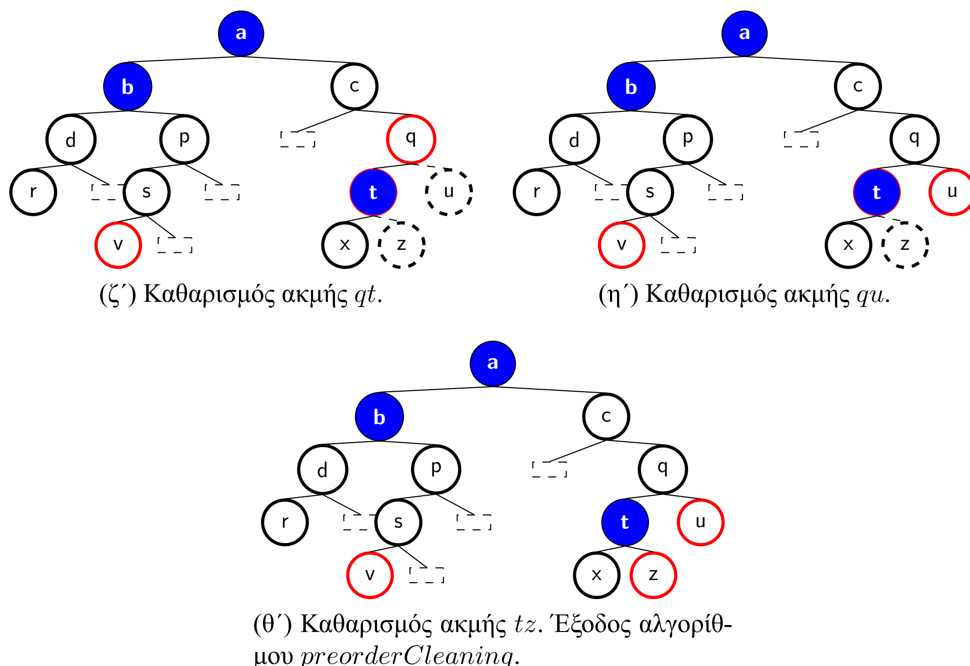
(στ΄) Καθαρισμός ακμής cq.

Σχήμα 6.1: Στιγμιότυπο αλγορίθμου $preorderCleaning$. (1ο Μέρος).

- *Επίσκεψη στον κόμβο* x (Σχήμα 6.1θ΄). Κάθε ακμή που προσπίπτει στον κόμβο x είναι καθαρή.

- *Επίσκεψη στον κόμβο* z (Σχήμα 6.1θ΄). Όλες οι ακμές που προσπίπτουν στον κόμβο z είναι καθαρές.

- *Επίσκεψη στον κόμβο* u (Σχήμα 6.1θ΄). Καμία μολυσμένη ακμή δεν προσπίπτει στον κόμβο u.



(ζ΄) Καθαρισμός ακμής qt.

(η΄) Καθαρισμός ακμής qu.

(θ΄) Καθαρισμός ακμής tz. Έξοδος αλγορίθμου $preorderCleaning$.

Σχήμα 6.1: Στιγμιότυπο αλγορίθμου $preorderCleaning$ (2ο Μέρος).

6.3 Ανάλυση αλγορίθμου

6.3.1 Απόδειξη ορθότητας

Λήμμα 6.3.1. *Έστω $u_1, u_2, \ldots, u_n$ μια ακολουθία από μη επισημασμένους κόμβους ενός τυχαίου δυαδικού δέντρου T τέτοιοι ώστε, 1. ο u_i είναι πατέρας του u_{i+1}, για κάθε $1 \leq i \leq n-1$ 2. ο u_i έχει 1 μόνο παιδί, για κάθε $1 \leq i \leq n-1$ 3. η ακμή uu_{i+1} είναι μολυσμένη, για κάθε $1 \leq i \leq n-1$ 4. μόνο στον κόμβο u_1 υπάρχει ερευνητής 5. στον κόμβο u_1, εκτός της ακμής u_1u_2, δεν προσπίπτει καμία άλλη μολυσμένη ακμή, τότε η επεξεργασία του κόμβου u_i, για κάθε $1 \leq i \leq n-1$, συνεπάγεται την ολίσθηση του ερευνητή του κόμβου u_i κατά μήκος της μολυσμένης ακμής u_iu_{i+1} και κατά συνέπεια τον καθαρισμό αυτής.*

Απόδειξη. Θα αποδείξουμε το λήμμα χρησιμοποιώντας τη μέθοδο της μαθηματικής επαγωγής:

- *Βάση της επαγωγής.* Το λήμμα ισχύει μετά την επεξεργασία του κόμβου u_1. Σε αυτόν, με βάση την υπόθεση του λήμματος, προσπίπτει μία μόνο μολυσμένη ακμή, η ακμή u_1u_2. Κατά συνέπεια, κατά την επεξεργασία του κόμβου u_1 η συνθήκη ελέγχου της 3ης γραμμής του αλγορίθμου είναι αληθής, οπότε ο ερευνητής απομακρύνεται από τον κόμβο u_1 ολισθαίνοντας κατά μήκος της μολυσμένης ακμής u_1u_2 (γραμμή 4), με αποτέλεσμα τον καθαρισμό αυτής (γραμμή 5).

- *Υποθέτουμε επαγωγικά* ότι το λήμμα ισχύει μετά από την επεξεργασία του κόμβου u_k, $k \leq n-2$, υποθέτουμε δηλαδή ότι μετά την επεξεργασία του κόμβου u_k, ισχύει

ότι ο ερευνητής του κόμβου u_k ολισθαίνει κατά μήκος της μολυσμένης ακμής $u_k u_{k+1}$ καθαρίζοντάς την.

- Θα αποδείξουμε ότι το λήμμα συνεχίζει να ισχύει και μετά την επεξεργασία του κόμβου u_{k+1}, θα αποδείξουμε δηλαδή ότι μετά την επεξεργασία του κόμβου u_{k+1} ο ερευνητής που βρισκόταν στον κόμβο u_{k+1} βρίσκεται πλέον στον κόμβο u_{k+2}, και η ακμή $u_{k+1}u_{k+2}$ είναι καθαρή. Από την υπόθεση του λήμματος έχουμε ότι η ακμή $u_k u_{k+1}$ είναι καθαρή, ενώ από την επαγωγική υπόθεση ισχύει ότι ο κόμβος u_{k+1} έχει 1 μόνο παιδί ενώ η ακμή που συνδέει τον κόμβο u_{k+1} με το παιδί του είναι μολυσμένη. Κατά συνέπεια, κατά την επεξεργασία του κόμβου u_{k+1} η συνθήκη ελέγχου της 3ης γραμμής του αλγορίθμου είναι αληθής, οπότε ο ερευνητής απομακρύνεται από τον κόμβο u_{k+1} ολισθαίνοντας κατά μήκος της μολυσμένης ακμής $u_{k+1}u_{k+2}$ (γραμμή 5), με αποτέλεσμα τον καθαρισμό αυτής (γραμμή 6). □

Λήμμα 6.3.2. *Ο αλγόριθμος $preorderCleaning$ καθαρίζει όλες τις ακμές του δυαδικού δέντρου που ανήκουν στο δεξιό υποδέντρο του πλησιέστερου επισημασμένου προγόνου τους.*

Απόδειξη. Θα αποδείξουμε το λήμμα χρησιμοποιώντας τη μέθοδο της μαθηματικής επαγωγής.

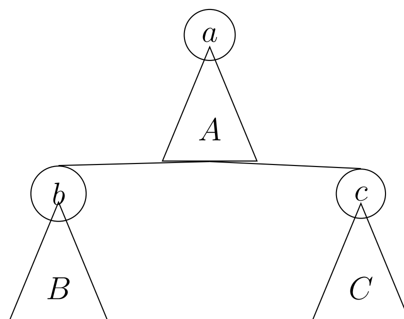
- *Βάση της επαγωγής.* Για δυαδικό δέντρο με 1 μόνο επισημασμένο κόμβο το λήμμα ισχύει.

- *Υποθέτουμε επαγωγικά* ότι το λήμμα ισχύει για κάθε δυαδικό δέντρο με k επισημασμένους κόμβους.

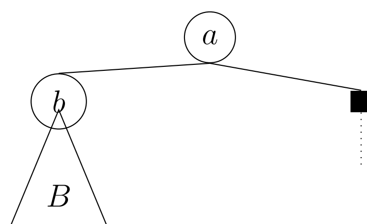
- Θα αποδείξουμε ότι το λήμμα ισχύει και για κάθε δυαδικό δέντρο με $k+1$ επισημασμένους κόμβους. Έστω δυαδικό δέντρο T αποτελούμενο από n κόμβους, όπου A, B και C υποδέντρα του T τέτοια ώστε οι επισημασμένοι κόμβοι a, b και c είναι ρίζες των υποδέντρων A, B και C αντίστοιχα, και $1, x \leq k, y \leq k$ είναι το πλήθος των επισημασμένω κόμβων των υποδέντρων A, B και C αντίστοιχα (Σχήμα 6.2).

  1. Ο 1ος κόμβος που επεξεργάζεται ο αλγόριθμός μας είναι ο κόμβος a. Το τι θα συμβεί εξαρτάται από το πόσες μολυσμένες ακμές προσπίπτουν σε αυτόν. Αυτό που για την ώρα μπορούμε να πούμε με σιγουριά είναι ότι, με βάση το λήμμα 5.3.2, κάθε ακμή του υποδέντρου A που ανήκει στο αριστερό υποδέντρο του κόμβου a είναι καθαρή. Κατά συνέπεια, η ακμή που ενώνει τον κόμβο a με το αριστερό του παιδί είναι καθαρή. Το ίδιο ισχύει και για την ακμή που ενώνει τον κόμβο b με τον πατέρα του, όποιος και αν είναι αυτός. Έτσι, οι ενέργειες του αλγορίθμου μας κατά την επεξεργασία του κόμβου a εξαρτώνται από τη δομή του δεξιού υποδέντρου του κόμβου a.

  2. Όσον αφορά στο αριστερό υποδέντρο του κόμβου a διακρίνουμε τις ακόλουθες περιπτώσεις:

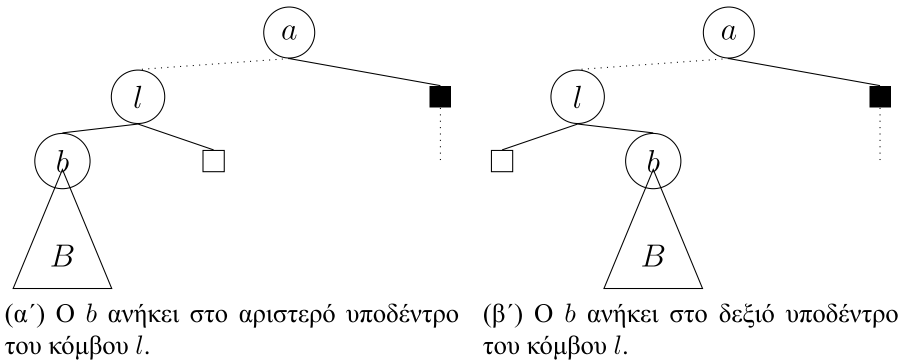


Σχήμα 6.2



Σχήμα 6.3: Ο a είναι ο πατέρας του b.

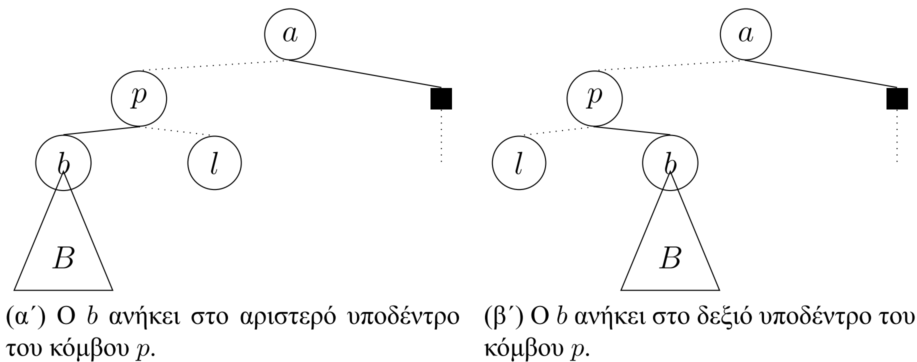
- *Έστω a ο πατέρας του κόμβου b (Σχήμα 6.3).* Αφού τελειώσει με τον κόμβο a, ο αλγόριθμός μας καλεί αναδρομικά τον εαυτό του για το αριστερό του παιδί (γραμμή 9), στην προκειμένη περίπτωση τη ρίζα του υποδέντρου B (κόμβος b). Όμως, με βάση την επαγωγική μας υπόθεση, το λήμμα ισχύει για το υποδέντρο B, αφού αυτό αποτελείται από το πολύ k επισημασμένους κόμβους. Έτσι, μετά την επεξεργασία του δεξιότερου κόμβου του υποδέντρου B, από ένας ερευνητής βρίσκεται σε κάθε επισημασμένο κόμβο του B, ενώ κάθε ακμή του υποδέντρου B που ανήκει στο δεξιό υποδέντρο του πλησιέστερου επισημασμένου προγόνου της είναι καθαρή. Κατά συνέπεια, όλες οι ακμές του υποδέντρου B είναι καθαρές, και άρα όλες οι ακμές που ανήκουν στο αριστερό υποδέντρο του κόμβου a είναι καθαρές.
- *Έστω l το αριστερότερο φύλλο του υποδέντρου A.*
  * *Ο l είναι πατέρας του κόμβου b (Σχήμα 6.4).* Τότε, μετά το κόμβο a, ο αλγόριθμός μας επεξεργάζεται με τη σειρά, όλους τους κόμβους του υποδέντρου A, ξεκινώντας από το αριστερό παιδί του κόμβου a και τελειώνοντας με τον κόμβο l. Άφου σε αυτούς τους κόμβους προσπίπτουν μόνο καθαρές ακμές, η συνθήκη ελέγχου της 3ης γραμμής του αλγορίθμου θα είναι ψευδής. Έπειτα, ο αλγόριθμος καλεί αναδρομικά τον εαυτό του, πρώτα για το αριστερό (γραμμή 9) και ύστερα για το δεξί παιδί του κόμβου l (γραμμή 10). Το ένα από αυτά θα είναι κενό, ενώ το άλλο θα είναι η ρίζα του υποδέντρου B, δηλαδή ο κόμβος b, για τον οποίο θα ισχύει ότι και στην προηγούμενη περίπτωση. Και σε αυτήν την περίπτωση λοιπόν, όλες οι ακμές που ανήκουν στο αριστερό υποδέντρο του κόμβου a είναι καθαρές.



(α΄) Ο b ανήκει στο αριστερό υποδέντρο του κόμβου l.

(β΄) Ο b ανήκει στο δεξιό υποδέντρο του κόμβου l.

Σχήμα 6.4: Ο l είναι πατέρας του b.

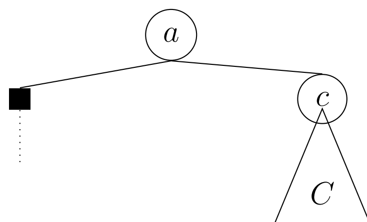


(α΄) Ο b ανήκει στο αριστερό υποδέντρο του κόμβου p.

(β΄) Ο b ανήκει στο δεξιό υποδέντρο του κόμβου p.

Σχήμα 6.5: Ο l δεν είναι πατέρας του b. Έστω p ο πατέρας του b.

* *Ο l δεν είναι πατέρας του κόμβου b (Σχήμα 6.5)* Τότε, έστω p ο πατέρας του κόμβου b.
  · *Ο b ανήκει στο αριστερό υποδέντρο του κόμβου p (Σχήμα 6.5α΄).* Ο αλγόριθμός μας ξεκινά τη λειτουργία του με την επεξεργασία του κόμβου a. Τελειώνοντας μαζί του, επεξεργάζεται εκείνους τους κόμβους του υποδέντρου A που ανήκουν στο αριστερό υποδέντρο του κόμβου a και είναι πρόγονοι του κόμβου b. Η επεξεργασία ξεκινά από το αριστερό παιδί του κόμβου a και καταλήγει στον κόμβο p. Σε όλους προσπίπτουν μόνο καθαρές ακμές, οπότε η συνθήκη ελέγχου της 3ης γραμμής είναι πάντα ψευδής. Ακολούθως, ο αλγόριθμος καλεί αναδρομικά τον εαυτό του πρώτα για το αριστερό (γραμμή 9) και ύστερα για το δεξί παιδί του κόμβου p (γραμμή 10). Για το αριστερό παιδί του κόμβου p, δηλαδή για τον κόμβο b, ισχυεί ότι και στις προηγούμενες δύο περιπτώσεις. Το δεξί παιδί του κόμβου p, όπως και κάθε απόγονος αυτού ανήκει στο υποδέντρο A στο αριστερό υποδέντρο του κόμβου a. Όμως σε κάθε τέτοιον κόμβο, με βάση το λήμμα 5.3.2, προσπίπτουν μόνο καθαρές ακμές, οπότε η συνθήκη ελέγχου της 3ης γραμμής του αλγορίθμου θα είναι ψευδής. Και πάλι, όλες



Σχήμα 6.6: Ο a είναι ο πατέρας του c.

οι ακμές που ανήκουν στο αριστερό υποδέντρο του κόμβου a είναι καθαρές.

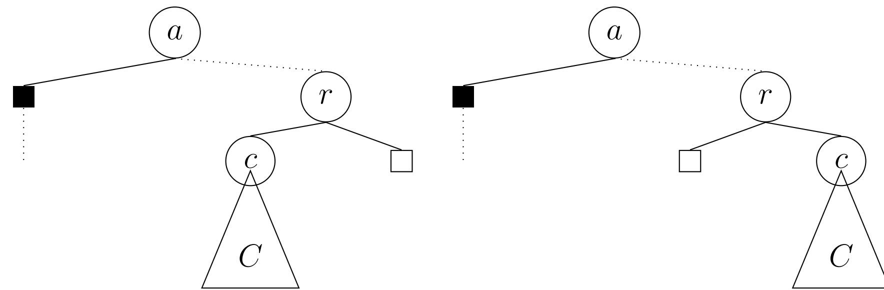
· *Ο b ανήκει στο δεξιό υποδέντρο του κόμβου p (Σχήμα 6.5β΄).* Η απόδειξη είναι εντελώς συμμετρική με αυτήν της προηγούμενης περίπτωσης.

- Συνεχίζουμε με το δεξιό υποδέντρο του κόμβου a, όπου διακρίνουμε τις ακόλουθες περιπτώσεις:

  – *Έστω a ο πατέρας του c (Σχήμα 6.6).* Τότε, με βάση το λήμμα 5.3.2, κάθε ακμή που προσπίπτει στον κόμβο a είναι καθαρή. Έτσι, καθώς ο αλγόριθμος επεξεργάζεται το συγκεκριμένο κόμβο, η συνθήκη ελέγχου της 3ης του γραμμής είναι ψευδής. Ακολουθεί η επεξεργασία του αριστερού υποδέντρου του κόμβου a, δηλάδή του υποδέντρου B, η οποία όπως είδαμε προηγουμένως οδηγεί πάντα σε καθαρισμό όλων των ακμών που ανήκουν στο αριστερό υποδέντρο του κόμβου a. Έπειτα, ο αλγόριθμος καλεί αναδρομικά τον εαυτό του με είσοδο τη ρίζα του υποδέντρου C, το οποίο αποτελείται από το πολύ k επισημασμένους κόμβους. Όμως, με βάση την επαγωγική μας υπόθεση, το λήμμα ισχύει για κάθε δυαδικό δέντρο που αποτελείται από $\leq k$ επισημασμένους κόμβους. Έτσι, όλες οι ακμές του υποδέντρου C που ανήκουν στο δεξιό υποδέντρο του πλησιέστερου επισημασμένου προγόνου τους είναι καθαρές, οπότε όλες οι ακμές του υποδέντρου C είναι καθαρές, και άρα όλες οι ακμές του δυαδικού δέντρου είναι πλέον καθαρές.

  – *Έστω r το δεξιότερο φύλλο του υποδέντρου A.*

    * *Ο r είναι πατέρας του κόμβου c (Σχήμα 6.7).* Τότε, η μοναδική μολυσμένη ακμή που προσπίπτει στον κόμβο a είναι η ακμή που οδηγεί στο δεξί του παιδί του. Έτσι, κατά την επεξεργασία του κόμβου a, η συνθήκη ελέγχου της 3ης γραμμής του αλγορίθμου είναι αληθής, οπότε ο ερευνητής απομακρύνεται από τον κόμβο a ολισθαίνοντας κατά μήκος της μοναδικής μολυσμένης ακμής του (γραμμή 4), η οποία λόγω της ολίσθησης καθαρίζει (γραμμή 5). Αν ο κόμβος a είναι πατέρας του κόμβου r, τότε, η άφιξη του ερευνητή στον κόμβο r συνεπάγεται την ύπαρξη ενός ερευνητή σε κάθε άκρο της ακμής rc. Κατά συνέπεια, η συνθήκη ελέγχου της 6ης γραμμής του αλγορίθμου είναι αληθής, οπότε η ακμή rc καθαρίζει (γραμμή 7). Από την άλλη, αν ο κόμβος a δεν είναι πατέρας του κόμβου r, τότε η συνθήκη



(α΄) Ο c ανήκει στο αριστερό υποδέντρο του κόμβου a.

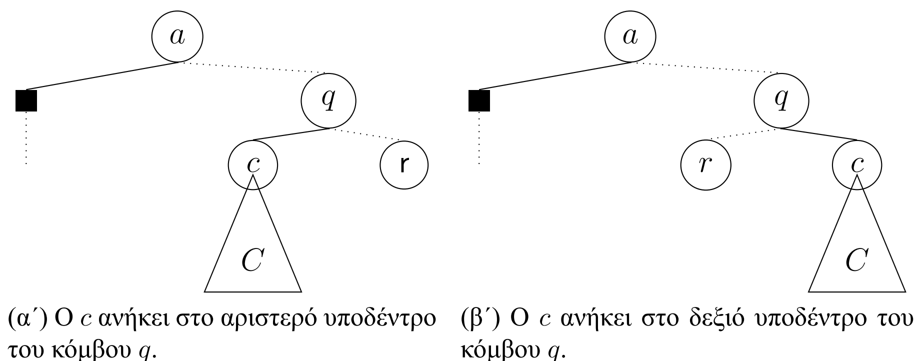
(β΄) Ο c ανήκει στο δεξιό υποδέντρο του κόμβου a.

Σχήμα 6.7: Ο r είναι πατέρας του c.

ελέγχου της 6ης γραμμής του αλγορίθμου είναι ψευδής. Ακολούθως, ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για το αριστερό παιδί του κόμβου a (γραμμή 8), και αφού επεξεργαστεί κάθε κόμβο που ανήκει στο αριστερό υποδέντρο του κόμβου a, καλεί αναδρομικά τον εαυτό του για το δεξί παιδί του κόμβου a (γραμμή 9). Αν ο κόμβος a είναι πατέρας του κόμβου r, τότε κάθε ακμή που προσπίπτει στον κόμβο r είναι καθαρή, οπότε η συνθήκη ελέγχου της 3ης γραμμής του αλγορίθμου είναι ψευδής. Αν όμως ο κόμβος a δεν είναι πατέρας του κόμβου r, τότε, με βάση το λήμμα 5.3.1, μετά την επεξεργασία του πατέρα του κόμβου r, ο ερευνητής του δεξιού παιδιού του κόμβου a καταλήγει στον κόμβο r, ενώ κάθε ακμή μεταξύ των κόμβων a και r καθαρίζει. Επιπλέον, τη στιγμή που ο αλγόριθμος επεξεργάζεται τον πατέρα του r, η άφιξη του ερευνητή στον κόμβο r συνεπάγεται την ύπαρξη ενός ερευνητή σε κάθε άκρο της ακμής rc. Κατά συνέπεια, η συνθήκη ελέγχου της 6ης γραμμής του αλγορίθμου είναι αληθής, οπότε η ακμή rc καθαρίζει (γραμμή 7). Έπειτα, ο αλγόριθμος καλεί αναδρομικά τον εαυτό του, πρώτα για το αριστερό (γραμμή 9) και ύστερα για το δεξί παιδί του πατέρα του κόμβου r (γραμμή 10). Το ένα από τα δύο θα είναι κενό, ενώ το άλλο θα είναι προφανώς ο κόμβος r. Κατά την επεξεργασία του κόμβου r, αφού σε αυτόν προσπίπτουν μόνο καθαρές ακμές, η συνθήκη ελέγχου της 3ης γραμμής του αλγορίθμου θα είναι ψευδής, οπότε ο αλγόριθμος καλεί αναδρομικά τον εαυτό του, πρώτα για το αριστερό (γραμμή 9) και ύστερα για το δεξί παιδί του κόμβου r (γραμμή 10). Το ένα από τα δύο θα είναι κενό, ενώ το άλλο θα είναι η ρίζα του υποδέντρου C, για το οποίο ισχύει ότι και στην προηγούμενη περίπτωση. Και σε αυτήν την περίπτωση λοιπόν όλες οι ακμές που ανήκουν στο υποδέντρο C καθαρίζουν, και άρα όλες οι ακμές του δέντρου είναι καθαρές.

* *Ο r δεν είναι πατέρας του κόμβου c (Σχήμα 6.8)* Τότε, έστω q ο πατέρας του κόμβου c.

  · *Ο c ανήκει στο αριστερό υποδέντρο του κόμβου q. (Σχήμα 6.8α΄)*. Τότε, η μοναδική μολυσμένη ακμή που προσπίπτει στον κόμβο a είναι η ακμή



(α΄) O c ανήκει στο αριστερό υποδέντρο του κόμβου q.

(β΄) O c ανήκει στο δεξιό υποδέντρο του κόμβου q.

Σχήμα 6.8: O r δεν είναι πατέρας του c. Έστω q ο πατέρας του c.

που οδηγεί στο δεξί του παιδί. Έτσι, κατά την επεξεργασία του κόμβου a η συνθήκη ελέγχου της 3ης γραμμής του αλγορίθμου είναι αληθής, οπότε ο ερευνητής απομακρύνεται από τον κόμβο a ολισθαίνοντας κατά μήκος της της μοναδικής μολυσμένης ακμής του (γραμμή 4), η οποία λόγω της ολίσθησης καθαρίζει (γραμμή 5). Αν ο κόμβος a είναι πατέρας του κόμβου q, τότε η άφιξη του ερευνητή στον κόμβο q συνεπάγεται την ύπαρξη ενός ερευνητή σε κάθε άκρο της ακμής qc. Κατά συνέπεια, η συνθήκη ελέγχου της 6ης γραμμής του αλγορίθμου είναι αληθής, οπότε η ακμή qc καθαρίζει (γραμμή 7). Αν όμως ο κόμβος a δεν είναι πατέρας του κόμβου q, τότε, με βάση το λήμμα 6.3.1, μετά την επεξεργασία του πατέρα του κόμβου q, ο ερευνητής του δεξιού παιδιού του κόμβου a καταλήγει στον κόμβο q, ενώ κάθε ακμή μεταξύ των κόμβων a και q καθαρίζει. Επιπλέον, τη στιγμή που ο αλγόριθμος επεξεργάζεται τον πατέρα του q, η άφιξη του ερευνητή στον κόμβο q συνεπάγεται την ύπαρξη ενός ερευνητή σε κάθε άκρο της ακμής qc. Κατά συνέπεια, η συνθήκη ελέγχου της 6ης γραμμής του αλγορίθμου είναι αληθής, οπότε η ακμή qc καθαρίζει (γραμμή 7). Έπειτα, ο αλγόριθμος καλεί αναδρομικά τον εαυτό του, πρώτα για το αριστερό (γραμμή 9) και ύστερα για το δεξί παιδί του πατέρα του κόμβου q (γραμμή 10). Το ένα από τα δύο θα είναι κενό, ενώ το άλλο θα είναι προφανώς ο κόμβος q. Κατά την επεξεργασία του κόμβου q, η μοναδική μολυσμένη ακμή που προσπίπτει σε αυτόν είναι αυτή που οδηγεί στο δεξί του παιδί. Κατά συνέπεια, ο ερευνητής του κόμβου q ολισθαίνει κατά μήκος αυτής (γραμμή 4) και την καθαρίζει (γραμμή 5). Έπειτα, ο αλγόριθμος καλεί αναδρομικά τον εαυτό του, πρώτα για το αριστερό (γραμμή 9) και ύστερα για το δεξί παιδί του κόμβου q (γραμμή 10). Αριστερό παιδί του κόμβου q είναι η ρίζα του υποδέντρου C, δηλαδή ο κόμβος c, για τον οποίο ισχύει ότι και στις προηγούμενες δύο περιπτώσεις. Κατά την επεξεργασία του δεξιού παιδιού του κόμβου q, αν ο q είναι πατέρας του κόμβου r, τότε στον r προσπίπτουν μόνο καθαρές ακμές, οπότε η συν-

θήκη της 3ης γραμμής του αλγορίθμου είναι ψευδής. Αν από την άλλη ο κόμβος q δεν είναι πατέρας του r, τότε, με βάση το λήμμα 6.3.1, μετά την επεξεργασία του πατέρα του κόμβου r, ο ερευνητής του κόμβου q καταλήγει στον κόμβο r, ενώ κάθε ακμή μεταξύ των κόμβων q και r καθαρίζει. Στη συνέχεια, ο αλγόριθμος καλεί αναδρομικά τον εαυτό του, πρώτα για το αριστερό (γραμμή 9) και ύστερα για το δεξί παιδί του πατέρα του κόμβου r (γραμμή 10). Το ένα από τα δύο θα είναι κενό, ενώ το άλλο θα είναι προφανώς ο κόμβος r. Κατά την επεξεργασία του κόμβου r, η συνθήκη ελέγχου της 3ης γραμμής του αλγορίθμου είναι ψευδής, αφού στον κόμβο r προσπίπτουν μόνο καθαρές ακμές. Έτσι, ο αλγόριθμος καλεί αναδρομικά τον εαυτό του, πρώτα για το αριστερό (γραμμή 9) και ύστερα για το δεξί παιδί του κόμβου r (γραμμή 10). Ο r όμως δεν έχει παιδιά. Και πάλι, όλες οι ακμές που ανήκουν στο υποδέντρο C καθαρίζουν, και άρα όλες οι ακμές του δέντρου είναι καθαρές.

· *Ο c ανήκει στο δεξιό υποδέντρο του κόμβου q. (Σχήμα 6.8β΄).* Η απόδειξη είναι εντελώς συμμετρική. □

6.3.2 Απόδειξη πολυπλοκότητας

Λήμμα 6.3.3. *Η κλήση $preorderCleaning(u)$, όπου u η ρίζα ενός τυχαίου δυαδικού δέντρου T με n κόμβους, απαιτεί χρόνο $\Theta(n)$.*

Απόδειξη. Αρκεί να αποδείξουμε ότι ο χρόνος d που χρειάζεται ο αλγόριθμος μας για να επεξεργαστεί 1 μόνο κόμβο (γραμμές $5-9$) είναι σταθερός και ανεξάρτητος του μεγέθους της εισόδου n. Ο παρακάτω πίνακας συνοψίζει το κόστος εκτελέσεως και το πλήθος εκτελέσεων κάθε γραμμής κατά τη διάρκεια της επεξεργασίας ενός κόμβου u από τον αλγόριθμο $preorderCleaning$:

Γραμμή	*Κόστος*	*Αριθμός εκτελέσεων*
3	c_3	1
4	c_4	≤ 1
5	c_5	≤ 1
6	c_6	≤ 1
7	c_7	≤ 1

$$d \leq c_3 + c_4 + c_5 + c_6 + c_7$$

Αυτός ο χρόνος εκτέλεσης μπορεί να γραφεί στη μορφή a όπου η σταθερά a εξαρτάται από τα διάφορα κόστη c_i των εντολών. Επομένως, είναι μια σταθερή συνάρτηση του n. □

6.4 Άλλα 4 σημαντικά λήμματα

Λήμμα 6.4.1. *Από τα λήμματα 4.3.5, 5.3.2 και 6.3.2 συνεπάγεται ότι, αν k το πλήθος των επισημασμένων κόμβων ενός τυχαίο δυαδικού δέντρου T, όλες οι ακμές του οποίου είναι*

μολυσμένες, τότε ο αλγόριθμός μας καθαρίζει το T χρησιμοποιώντας k ερευνητές.

Λήμμα 6.4.2. *Από τα λήμματα 4.4.1 και 6.4.1, συνεπάγεται ότι ο αλγόριθμός μας είναι βέλτιστος.*

Λήμμα 6.4.3. *Από το λήμμα 6.4.2, συνεπάγεται ότι ο αλγόριθμος $putSearchers$ υπολογίζει το $monotone\ exclusive\ search\ number$ ενός τυχαίου δυαδικού δέντρου T.*

Η χρονική πολυπλοκότητα του αλγορίθμου μας ισούται με το άθροισμα της χρονικής πολυπλοκότητας των αλγορίθμων $postorderLabeling$, $putSearchers$, $postorderCleaning$ και $preorderCleaning$. Κατά συνέπεια, ισχύει το ακόλουθο λήμμα:

Λήμμα 6.4.4. *Η χρονική πολυπλοκότητα του αλγορίθμου μας ισούται με $\Theta(n)$.*

Κεφάλαιο 7

Επεξήγηση κώδικα παραρτήματος

Πριν υλοποιήσουμε τους αλγορίθμους που περιγράψαμε στο προηγούμενο κεφάλαιο θα υλοποιήσουμε την είσοδό τους, δηλαδή ένα δυαδικό δέντρο. Υπάρχουν αρκετοί τρόποι για να το υλοποιήσουμε, οπότε εμείς θα διαλέξουμε έναν από αυτούς και θα το αποθηκεύσουμε σαν ένα δυαδικό δέντρο αναζήτησης, χωρίς αυτό να σημαίνει πως αν το αποθηκεύαμε κάπως αλλιώς δε θα έτρεχε ο αλγόριθμός μας.

7.1 Το αρχείο BST.h

Στις γραμμές $3 - 5$ ορίζονται 3 σταθερές $PARENT$, $LEFT_CHILD$ και $RIGHT_CHILD$, οι οποίες ισούνται με 0, 1 και 2 αντίστοιχα. Στη συνέχεια, στις γραμμές $6 - 14$, ορίζουμε τι είναι κόμβος. Αποθηκεύουμε λοιπόν κάθε κόμβο ως ένα $struct\ node$, το οποίο θα φέρει

- ένα κλειδί key, βάσει του οποίου αποθηκεύεται η πληροφορία.
- έναν ακέραιο $agent$, η τιμή του οποίου ισούται με
  - 1, αν ο κόμβος έχει ερευνητή.
  - 0, αλλιώς.
- έναν ακέραιο nof_cont_edges, ο οποίος αποθηκεύει το πλήθος των μολυσμένων ακμών που προσπίπτουν στον κόμβο.
- έναν ακέραιο $label$, που ισούται με
  - 1, αν ο κόμβος είναι επισημασμένος.
  - 0, αν ο κόμβος είναι μη επισημασμένος.
- έναν πίνακα $is_contaminated$ 3 θέσεων, η 1η εκ των οποίων κρατά πληροφοριά για την ακμή που ενώνει τον κόμβο με τον πατέρα του, ενώ η 2η και η 3η για την ακμή που ενώνει τον κόμβο με το αριστερό και το δεξί παιδί του αντίστοιχα. Κάθε τέτοιος ακέραιος ισούται με

  - 1, αν η αντίστοιχη ακμή είναι μολυσμένη,
  - 0, αν είναι καθαρή, και
  - -1, αν η ακμή δεν υπάρχει.

- έναν πίνακα 3 θέσεων, που περιέχει τους δείκτες προς του γειτονικούς του κόμβους. Ειδικότερα
  - η θέση 0 του πίνακα αποθηκεύει δείκτη προς τον πατέρα του κόμβου,
  - η θέση 1 αποθηκεύει δείκτη προς το αριστερό του παιδί, και
  - η θέση 2 δείκτη προς το δεξί παιδί του κόμβου.

Βάσει του $node$, ένα διασυνδεδεμένο δυαδικό δέντρο υλοποιείται με το $struct\ tree$, χρησιμοποιώντας έναν δείκτη $root$ προς τη ρίζα του δέντρου. Οι γραμμές που ακολουθούν περιλαμβάνουν τις δηλώσεις των συναρτήσεων που ορίζονται στο αρχείο $cleaning.c$.

7.2 Το αρχείο BST.c

- $get_root(T)$ Επιστρέφει το δείκτη προς τη ρίζα του δέντρου T.
- $get_data(u)$ Επιστρέφει την τιμή του πεδίου $data$ του κόμβου u.
- $get_agent(u)$ Επιστρέφει την τιμή του πεδίου $agent$ του κόμβου u.
- $get_label(u)$ Επιστρέφει την τιμή του πεδίου $label$ του κόμβου u.
- $get_nof_cont_edges(u)$ Επιστρέφει την τιμή του πεδίου nof_cont_edges του κόμβου u.
- $get_edge(u, i)$ Επιστρέφει δείκτη προς τον i-οστό γείτονα του κόμβου u.
- $get_is_contaminated(u, i)$ Επιστρέφει την τιμή της i-οστής θέσης του πεδιού-πίνακα $is_contaminated$ του κόμβου u.

Οι παραπάνω πράξεις αν και επιτρέπουν την επισκόπηση του δέντρου, απαγορεύουν οποιαδήποτε τροποποίηση του. Με ρική ευελιξία ως προς αυτήν την κατεύθυνση παρέχουν οι ακόλουθες πράξεις:

- $set_root(T, u)$ Θέτει τον κόμβο u ως τη ρίζα του δέντρου T.
- $set_data(u, val)$ Θέτει το val ως κλειδί του κόμβου u.
- $set_agent(u, val)$ Θέτει το val ως τιμή του πεδίου $agent$ του κόμβου u.
- $set_label(u, val)$ Θέτει το val ως τιμή του πεδίου $label$ του κόμβου u.
- $set_nof_cont_edges(u, val)$ Θέτει το val ως τιμή του πεδίου nof_cont_edges του κόμβου u.

- $set_edge(u, i, val)$ Θέτει τον κόμβο u ως τον i-οστό γείτονα του κόμβου v.

- $set_is_contaminated(u, i, val)$ Θέτει το val ως τιμή της i-οστής θέσης του πίνακα $is_contaminated$ του κόμβου u.

- $init(T)$ Αρχικοποιεί κάθε δυαδικό δέντρο T που δέχεται ως είσοδο, θέτοντας τη ρίζα του να δείχνει στο κενό.

- $creation(T)$ Δέχεται ως είσοδο ένα δυαδικό δέντρο T. Στη γραμμή 86, η συνάρτηση $init$ αρχικοποιεί το T. Σε ένα αρχείο $input.txt$ υπάρχουν, το ένα κάτω από το άλλο τα κλειδιά, βάσει των οποίων θα αποθηκεύσουμε τους κόμβους μας, καθώς και ο αριθμός -1, ο οποίος σηματοδοτεί το τέλος του αρχείου μας. Έτσι, στη γραμμή 88, η συνάρτηση $fopen$ ανοίγει το txt αρχείο για να διαβάσει το περιεχόμενό του. Ακολουθεί ένας επαναληπτικός βρόχος $do...while$ (γραμμές $90-95$), κατά την εκτέλεση του οποίου η $fscanf$ διαβάζει τον επόμενο ακέραιο που υπάρχει στο αρχείο (γραμμή 92) και τον αποθηκεύει σε μία μεταβλητή x. Αν ο x είναι διάφορος του -1 (γραμμή 93), τότε η συνάρτηση $insert$ εισάγει στο T ένα νέο κόμβο με τιμή κλειδιού, την τιμή της μεταβλητής x (γραμμή 94). Ο βρόχος επαναλαμβάνεται μέχρις ότου η $fscanf$ διαβάσει από το αρχείο την τίμή -1, η οποία δεν εισάγεται στο δέντρο.

- $isEmpty(T)$ Δέχεται ως είσοδο ένα δυαδικό δέντο T. Επιστρέφει 1 αν το δέντρο είναι κενό ($getRoot(T) = NULL$), διαφορετικά επιστρέφει 0.

- $newNode(num)$ Δεσμεύει την απαραίτητη μνήμη για ένα νέο κόμβο (γραμμή 110) και καταχωρίζει την τιμή της παραμέτρου της num, σαν τιμή κλειδιού του νέου κόμβου ($setKey(new, num)$). Αφού κάθε κόμβος του δέντρου θεωρείται αρχικά μη επισημασμένος, καταχωρίζει την τιμή 0 σαν τιμή του πεδίου $label$ του νέου κόμβου ($setLabel(new, 0)$), Αφού νέος κόμβος δεν έχει εισαχθεί ακόμη στο δέντρο, θέτει τους δείκτες προς κάθε γειτονικό κόμβο να δείχνουν στο κενό (γραμμές $115-116$). Κατά συνέπεια, σε κάθε θέση του πεδίου-πίνακα $is_contaminated$ θέτει την τιμή -1 (γραμμές $118-119$). Τέλος, καταχωρίζει την τιμή 0 σαν τιμή του πεδίου $nofContEdges$ του νέου κόμβου ($setNofContEdge(new, 0)$), και επιστρέφει ως τιμή τη διεύθυνση της θέσης μνήμης που δέσμευσε.

- $insert(T, num)$ Εισάγει το κλειδί num σε ένα ένα δυαδικό δέντρο T.
  - Η $newNode$ δημιουργεί ένα νέο κόμβο ptr με τιμή κλειδιού num και επιστρέφει τη διεύθυνσή του.
  - Αν το δέντρο είναι κενό, τότε η $setRoot$ απλώς αναθέτει στη ρίζα τη διεύθυνση του νέου κόμβου (γραμμή 133).
  - Σε περίπτωση που το δέντρο δεν είναι κενό, τότε η $getRoot$ επιστρέφει ως τιμή τη διεύθυνση της ρίζας του δέντρου (γραμμή 136), την οποία η $insert$ αφού αποθηκεύσει σε έναν κόμβο $current$, επαναλαμβάνει την ακόλουθη διαδικασία μέχρι να φτάσει σε κόμβο που να μην οδηγεί (ούτε αριστερά ούτε δεξιά) σε υποδέντρο. Θέτει ως τιμή του προηγούμενου κόμβου $previous$ τη διεύθυνση του

τρέχοντος κόμβου (γραμμή 139). Συγκρίνει την τιμή κλειδιού του νέου κόμβου με την τιμή κλειδιού του τρέχοντος κόμβου (γραμμή 140). Αν είναι μικρότερη, προχωρά στη ρίζα του αριστερού υποδέντρου (γραμμή 141), αφού σύμφωνα με την ιδιότητα του δυαδικού δέντρου αναζήτησης το num είναι αδύνατον να αποθηκευτεί στο δεξιό υποδέντρο. Αντίστοιχα, εάν η τιμή κλειδιού του νέου κόμβου είναι μεγαλύτερη από την τιμή κλειδιού του τρέχοντος κόμβου, προχωρά στη ρίζα του δεξιού υποδέντρου (γραμμή 143).

  - Στη συνέχεια, η $insert$ συγκρίνει την τιμή κλειδιού του νέου κόμβου με την τιμή κλειδιού του προηγούμενου κόμβου (γραμμή 146). Αν είναι μικρότερη, τότε η $insert_as_leaf$ εισάγει το νέο κόμβο ως αριστερό παιδί του προηγούμενου κόμβου (γραμμή 147), διαφορετικά ως δεξί παιδί του προηγούμενου κόμβου (γραμμή 149).

- $insertAsLeaf(prev, neos, i)$ Η συνάρτηση εισάγει ως

  - αριστερό παιδί, αν $i = 1$, ή
  - δεξί παιδί, αν $i = 2$,

  ενός κόμβου $prev$ ενός δυαδικού δέντρου, έναν νέο κομβο $neos$. Η τιμή του δείκτη προς το i-οστό παιδί του κόμβου $prev$ θα πρέπει να αλλάξει. Αυτό επιτυγχάνεται καταχωρίζοντας στην i-οστή θέση του πεδίου-πίνακα $edge$ του κόμβου $prev$ τη διεύθυνση του νέου κόμβου $neos$ (γραμμή 160). Θα πρέπει επίσης να ενήμερώσουμε τον κόμβο $prev$ ότι η ακμή που τον συνδέει με τον κόμβο $neos$ που μόλις προστέθηκε στο δέντρο είναι μολυσμένη αφού, πριν ξεκινήσει η διαδικασία του καθαρισμού, όλες οι ακμές του δέντρου είναι μολυσμένες. Αυτό επιτυγχάνεται καταχωρίζονατας την τιμή 1 στην i-οστή θέση του πεδίου-πίνακα $isContaminated$ του κόμβου $prev$ (γραμμή 161). Τελειώνοντας με τον κόμβο $prev$, θα πρέπει προφανώς να τον ενημερώσουμε ότι το πλήθος των μολυσμένων ακμών που προσπίπτουν σε αυτόν άλλαξε και ειδικότερα ότι αυξήθηκε κατα 1. Για να το πετύχουμε αυτό, αποθηκεύουμε σε μία μεταβλητή $contEdges$ την τιμή του πεδίου $nofContEdges$ του κόμβου $prev$ (γραμμή 162), αυξάνουμε κατά 1 την τιμή της (γραμμή 163) και την καταχωρούμε στο πεδίο $nofContEdges$ του κόμβου $prev$ (γραμμή 164). Στις γραμμές που ακολουθούν ενημερώνουμε τον κόμβο $neos$ για τις αλλαγές, τροποποιώντας τα αντίστοιχα πεδία του. Η συνάρτηση επιστρέφει ως τιμή τη διεύθυνση της θέσης μνήμης του κόμβου $neos$.

7.3 Το αρχείο placement.h

Περιλαμβάνει τις δηλώσεις των συναρτήσεων που ορίζονται στο αρχείο $placement.c$.

7.4 Το αρχείο placement.c

- $labeling(u, nofLnodes)$ Όπως σε κάθε αναδρομικό αλγόριθμο, έτσι και εδώ, θα πρέπει να υπάρχει μία αναδρομική περίπτωση. Αυτό συμβαίνει όταν η διεύθυνση

του κόμβου u είναι $NULL$. Αν δεν είναι $NULL$, ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για τη ρίζα πρώτα του αριστερού και ύστερα του δεξιού υποδέντρου του κόμβου u. Στη συνθήκη ελέγχου που ακολουθεί (γραμμή 13), η συνάρτηση $nofUnlabeledKids$ επιστρέφει το πλήθος των μη επισημασμένων παιδιών του κόμβου u, ενώ η συνάρτηση $getEdge$ επιστρέφει δείκτη προς τον πατέρα του u. Αν αυτός ο δείκτης είναι $NULL$, τότε ο κόμβος u είναι η ρίζα του δέντρου, αφού αυτή είναι ο μοναδικός κόμβος του δέντρου που δεν έχει πατέρα. Αν η συνθήκη ελέγχου είναι αληθής, τότε

- ο κόμβος u επισημαίνεται (γραμμή 15). Αυτό επιτυγχάνεται καταχωρίζοντας την τιμή 1 στο πεδίο $label$ του κόμβου u ($setLabel(u, 1)$).
- αυξάνεται κατά ένα η τιμή της μεταβλητής $nofLnodes$, η οποία σώζει το πλήθος των μη επισημασμένων παιδιών του δέντρου (γραμμή 16).

Τέλος (γραμμή 19), η συνάρτηση επιστρέφει την τιμή της μεταβλητής $nofLnodes$.

- $putSearchers(u, isLeft, nofSearchers)$ Όπως σε κάθε αναδρομικό αλγόριθμο, έτσι και εδώ, θα πρέπει να υπάρχει μία αναδρομική περίπτωση. Αυτό συμβαίνει όταν η διεύθυνση του κόμβου είναι $NULL$. Αν δεν είναι $NULL$, τότε η κλήση της συνάρτησης $isLabeled$ (γραμμή 26)

  - επιστρέφει 1 αν ο κόμβος u που δέχεται ως 1ο όρισμα είναι επισημασμένος. Σε αυτήν την περίπτωση, η ροή του προγράμματος μεταφέρεται στο σώμα του βρόχου if και ειδικότερα στη γραμμή 28, όπου λαμβάνουν χώρα 3 συγκρίσεις. Η 1η από αυτές ($getEdge(u, PARENT) == NULL$) είναι αληθής αν η τιμή του δείκτη προς τον πατέρα του u είναι $NULL$. Είναι δηλαδή αληθής αν ο κόμβος u είναι η ρίζα του δέντρου, αφού σε κάθε δέντρο η ρίζα είναι ο μοναδικός κόμβος που δεν έχει πατέρα. Η 2η σύγκριση ($getEdge(u, LEFT_CHILD) == NULL$) είναι αληθής αν η τιμή του δείκτη προς το αριστερό παιδί του κόμβου u είναι $NULL$, είναι δηλαδή αληθής αν ο κόμβος u δεν έχει αριστερό παιδί. Η 3η σύγκριση, τέλος ($getLabel(getEdge(u, LEFT_CHILD))$), είναι αληθής αν η συνάρτηση $getLabel$ με όρισμα δείκτη προς το αριστερό παιδί του κόμβου u επιστρέψει 1. Είναι δηλαδή αληθής αν το αριστερό παιδί του κόμβου u είναι ένας επισημασμένος κόμβος.

    * Αν λοιπόν είναι αληθείς η 1η σύγκριση και μία από τις άλλες 2, τότε ένας ερευνητής εισάγεται στη ρίζα του δέντρου (γραμμή 30). Αυτό επιτυγχάνεται καταχωρίζοντας την τιμή 1 στο πεδίο $agent$ του κόμβου u ($setAgent(u, 1)$). Κατά συνέπεια (γραμμή 31), αυξάνεται κατά 1 η τιμή της μεταβλητής $nofSearchers$, η οποία σώζει το πλήθος των ερευνητών που έχουν μέχρι τώρα εισαχθεί στο δέντρο.

  Στη γραμμή 33, η $putSearchers$ καλεί αναδρομικά τον εαυτό της με ορίσματα τη ρίζα του αριστερού υποδέντρου, την τιμή 1 και την τιμή της μεταβλητής $nofSearchers$, ενώ στη γραμμή 34 η $putSearchers$ καλεί αναδρομικά τον

εαυτό της με ορίσματα τη ρίζα του δεξιού υποδέντρου, την τιμή 0 και την τιμή της μεταβλητής $nofSearchers$.

– επιστρέφει 0 αν ο κόμβος u που δέχεται ως 1ο όρισμα είναι μη επισημασμένος. Τότε, η ροή του προγράμματος μεταφέρεται στο σώμα του βρόχου if και ειδικότερα στη γραμμή 28, όπου

  * αν ο κόμβος u δεν έχει μη επισημασμένα παιδιά ($nofUnlabeledKids == 0$) και η τιμή της μεταβλητής $isLeft$ ισούται με 1, τότε ένας ερευνητής εισάγεται στον κόμβο u και αυξάνεται κατά 1 η τιμή της μεταβλητής $nofSearchers$.

Στη γραμμή 43, η $putSearchers$ καλεί αναδρομικά τον εαυτό της με ορίσματα τη ρίζα του αριστερού υποδέντρου, την τιμή της μεταβλητής $isLeft$ και την τιμή της μεταβλητής $nofSearchers$, ενώ στη γραμμή 44 η $putSearchers$ καλεί αναδρομικά τον εαυτό της με ορίσματα τη ρίζα του δεξιού υποδέντρου, την τιμή της μεταβλητής $isLeft$ και την τιμή της μεταβλητής $nofSearchers$.

• $nofLabeledKids(u)$ Δέχεται ως είσοδο έναν κόμβο u ενός δυαδικού δέντρου και επιστρέφει το πλήθος των επισημασμένων παιδιών του. Στη γραμμή 52, μια μεταβλητή $labeledKids$ αρχικοποιείται στο 0. Η μεταβλητή αυτή σώζει το πλήθος των επισημασμένων παιδιών του κόμβου u που έχουν εντοπιστεί μέχρι τώρα. Στη γραμμή 55, η συνάρτηση $getEdge$, επιστρέφει σε μια μεταβλητή δείκτη v τη διεύθυνση του αριστερού παιδιού του κόμβου u. Στη γραμμή 56, αν η διεύθυνση του κόμβου v είναι ίση με την τιμή $NULL$, τότε ο κόμβος v δεν υπάρχει, δηλαδή ο κόμβος u δεν έχει αριστερό παιδί και η ροή του προγράμματος μεταφέρεται στη γραμμή 60. Αν η διεύθυνση του κόμβου v και η τιμή $NULL$ διαφέρουν, τότε ο κόμβος v υπάρχει και η συνθήκη ελέγχου που λαμβάνει χώρα στη γραμμή 56 είναι αληθής, οπότε η ροή του προγράμματος μεταφέρεται στη γραμμή 57. Εκεί πραγματοποιείται έλεγχος για το αν ο κόμβος v είναι επισημασμένος. Αυτό επιτυγχάνεται διαβάζοντας την τιμή του πεδίου $label$ του κόμβου v ($getEdge(v)$). Αν η $getEdge$ επιστρέψει 0 τότε ο κόμβος v είναι μη επισημασμένος και η συνθήκη ελέχου που λαμβάνει χώρα στη γραμμή 57 είναι ψευδής, οπότε η ροή του προγράμματος μεταφέρεται στη γραμμή 60. Αν όμως η συνάρτηση $getEdge$ επιστρέψει 1, τότε ο κόμβος v είναι επισημασμένος και η συνθήκη ελέχου που λαμβάνει χωρα στη γραμμή 57 είναι αληθής, οπότε η ροή του προγράμματος μεταφέρεται στο σώμα του if, όπου αυξάνεται κατά 1 η τιμή της μεταβλητής $labeledKids$. Με τον ίδιο ακριβώς τρόπο, στις γραμμές $60-63$, ελέγχεται αν το αριστερό παιδί του κόμβου u είναι επισημασμένο ή όχι. Κατά συνέπεια οι 4 αυτές γραμμές δε θα αναλυθούν περισσότερο. Η γραμμή 65, τέλος, επιστρέφει την τιμή της μεταβλητής $labeledKids$.

• $nofUnlabeledKids(u)$ Δέχεται ως είσοδο τη διεύθυνση ενός κόμβου u ενός δυαδικού δέντρου και επιστρέφει το πλήθος των μη επισημασμένων παιδιών του. Η λειτουργία της είναι εντελώς συμμετρική με αυτήν της συνάρτησης $nofLabeledKids$ και συνεπώς δε χρήζει ιδιαίτερης ανάλυσης.

- $isLabeled(u)$ Δέχεται ως είσοδο τη διεύθυνση ενός κόμβου u ενός δυαδικού δέντρου και επιστρέφει 1 αν ο u είναι επισημασμένος, αλλιώς επιστρέφει 0. Αν η διεύθυνση του κόμβου u είναι ίση με την τιμή $NULL$, τότε ο κόμβος u δεν υπάρχει και η 1η σύγκριση που λαμβάνει χώρα στη γραμμή 88 είναι ψευδής, κατά συνέπεια η συνάρτηση επιστρέφει την τιμή 0. Αν ωστόσο η διεύθυνση του κόμβου u δεν είναι $NULL$, τότε ο κόμβος u υπάρχει και η 1η σύγκριση της 88ης γραμμής του αλγορίθμου είναι αληθής, οπότε λαμβάνει χώρα μία 2η σύγκριση όπου η τιμή που επιστρέφει η συνάρτηση $geLabel$ συγκρίνεται με την τιμή 1. Αν λοιπόν η $getLabel$ επιστρέψει 1, τότε ο κόμβος u είναι επισημασμένος και η 2η σύγκριση της γραμμής 88 είναι αληθής, οπότε η συνάρτηση επιστρεφεί την τιμή 1. Διαφορετικά, ο κόμβος u είναι μη επισημασμένος και η 2η σύγκριση είναι ψευδής, οπότε η συνάρτηση επιστρεφεί την τιμή 0.

7.5 Το αρχείο cleaning.h

Περιλαμβάνει τις δηλώσεις των συναρτήσεων που ορίζονται στο αρχείο $cleaning.c.$

7.6 Το αρχείο cleaning.c

- $postorder_cleaning(u)$ Όπως σε κάθε αναδρομικό αλγόριθμο, έτσι και εδώ, θα πρέπει να υπάρχει μία αναδρομική περίπτωση. Αυτό συμβαίνει όταν η διεύθυνση του κόμβου u είναι $NULL$. Αν δεν είναι $NULL$, ο αλγόριθμος καλεί αναδρομικά τον εαυτό του για τη ρίζα πρώτα του αριστερού και ύστερα του δεξιού υποδέντρου του κόμβου u. Στη γραμμή 16, λαμβάνουν χώρα 2 συγκρίσεις. Η 1η ($get_label(u) == 0$) είναι αληθής όταν η get_label επιστρέφει 0, δηλαδή όταν ο κόμβος u ειναι μη επισημασμένος, ενώ η 2η όταν η συνάρτηση get_agent επιστρέφει 1, δηλαδή όταν ο κόμβος u έχει ερευνητή. Όταν λοιπόν ο u είναι μη επισημασμένος και έχει ερευνητή, τότε εκτελούνται οι εντολές στο σώμα του if (γραμμές $18 - 26$). Η $before_slide$ (γραμμή 18) επιστρέφει σε μια μεταβλητή $cont_edge$ τη σταθερά που χαρακτηρίζει τον κόμβο v, όπου uv η μοναδική μολυσμένη ακμή που προσπίπτει στον κόμβο u. Η κλήση της συνάρτησης $slide$ (γραμμή 19) συνεπάγεται την ολίσθηση του ερευνητή του κόμβου u κατά μήκος της ακμής uv. Έπειτα, ανάλογα με το αν ο u είναι αριστερό ή δεξί παιδί του κόμβου v, η $after_slide$ καθαρίζει την αντίστοιχη ακμή. Τέλος, η $searching_neighboring_nodes$ καθαρίζει κάθε μολυσμένη ακμή vw, όπου w κόμβος με ερευνητή.

- $preorder_cleaning(u)$ Οι λεπτομέρειες αυτής της συνάρτησης είναι παρόμοιες με αυτές της postorder_cleaning.

- $searching_of_neighboring_nodes(u)$ Δέχεται ως είσοδο δείκτη προς έναν κόμβο u και καθαρίζει κάθε μολυσμένη ακμή uv αν και μόνο αν σε κάθε της άκρο υπάρχει ερευνητής. Κάθε κόμβος ενός δυαδικού δέντρου έχει το πολύ 3 γειτονικούς κόμβους. Έτσι, στη γραμμή 74, λαμβάνει χώρα ένας βρόχος, ο οποίος επαναλαμβάνεται 3 φορές. Στην 1η του επανάληψη ($k = 0$) ελέγχει τον πατέρα του u, στη 2η το αριστερό

παιδί του u και στην3η το δεξί παιδί του u. Στη γραμμή 75 λαμβάνουν χώρα 2 συγκρίσεις η συνάρτηση get_edge μας επιστρέφει δείκτη προς τον k-οστό γείτονα του κόμβου u. Αν λοιπόν αυτός ο γείτονας υπάρχει ($get_edge(u, k) \neq NULL$), τότε λαμβάνει χώρα μία 2η σύγκριση ($get_agent(get_edge(u, k)) == 1$). Αν και αυτή η σύγκριση είναι αληθής, τότε ένας ερευνητής θα βρίσκεται στον k-οστό γείτονα του κόμβου u και η ροή του προγράμματος μεταφέρεται στο σώμα του βρόχου if. Στη γραμμή 77, η συνάρτηση $cleaning_edge$, καταχωρίζοντας κατάλληλες τιμές σε κατάλληλα πεδία του κόμβου u, τον ενημερώνει ότι η ακμή που οδηγεί στον k-οστό του γείτονα παύει να είναι μολυσμένη. Έπειτα, η συνάρτηση $cleaning_edge$ θα πρέπει να κάνει το ίδιο και με τον k-οστό γείτονα του κόμβου u, θα πρέπει δηλαδή να τον ενημερώσει ότι η ακμή που οδηγεί προς τον κόμβο u είναι καθαρή.

- $cleaning(u, i)$ Καθαρίζει εκείνη τη μολυσμένη ακμή, το ένα άκρο της οποίας προσπίπτει στον κόμβο u και το άλλο της άκρο στον κόμβο v, όπου v ο i-οστός γείτονας του κόμβου u. Περιλαμβάνει
  - την ενημέρωση του κόμβου u ότι η ακμή uv παύει να είναι μολυσμένη (γραμμή 99). Αυτό επιτυγχάνεται καταχωρίζοντας την τιμή 0 στην i-οστή θέση του πεδίου-πίνακα $isContaminated$ του κόμβου u ($setIsContaminated(u, i, 0)$).
  - τη μείωση κατά 1 του πλήθους των μολυσμένων ακμών που προσπίπτουν στον κόμβο u (γραμμές $100 - 102$) ως εξής
    * αποθήκευση σε μία μεταβλητή $contEdges$ του μέχρι τώρα πλήθους μολυσμένων ακμών που προσπίπτουν στον κόμβο u (γραμμή 100). Αυτό επιτυγχάνεται καταχωρίζοντας στη μεταβλητή $contEdges$ την τιμή του πεδίου $nofContEdges$ του κόμβου u ($nofContEdges = getNofContEdges(u)$).
    * μειώση κατά 1 της τιμής της μεταβλητής $contEdges$ (γραμμή 101)
    * αποθήκευση της τιμής της μεταβλητής $contEdges$ στο πεδίο $nofContEdges$ του κόμβου u (γραμμή 102).

- $before_slide(u)$ Καλείται με είσοδο τη διεύθυνση ενός κόμβου u, στον οποίο προσπίπτουν το πολύ 2 καθαρές ακμές και ακρίβως 1 μολυσμένη. Καλεί τη συνάρτηση $find_the_contaminated_edge$, η οποία της επιστρέφει τη σταθερά που χαρακτηρίζει τον κόμβο v, όπου uv η μολυσμένη ακμή που προσπίπτει στον κόμβο u. Η $before_slide$ επιστρέφει με τη σειρά της αυτή τη σταθερά στην καλούσα συνάρτηση.

- $after_slide(u, i, v, j)$ Η συνάρτηση αυτή δέχεται ως είσοδο 2 κόμβους u, v και 2 ακέραιους i, j, όπου v ο i-οστός γείτονας του κόμβου u και u ο j-οστός γείτονας του κόμβου v. Η κλήση της γίνεται αμέσως μόλις ένας ερευνητής ολισθήσει κατά μήκος της ακμής uv και συνεπάγεται
  - την ενημέρωση, με τη βοήθεια της συνάρτησης $cleaning_edge$, του κόμβου u ότι η ακμή προς τον i-οστό του γείτονα (κόμβος v) είναι καθαρή.

– την ενημέρωση, με τη βοήθεια της συνάρτησης $cleaning_edge$, του κόμβου v ότι η ακμή προς τον j-οστό του γείτονα (κόμβος u) είναι καθαρή.

- $findTheContaminatedEdge(u)$ Καλείται με είσοδο τη διεύθυνση ενός κόμβου u, στον οποίο προσπίπτουν το πολύ 2 καθαρές ακμές και ακρίβως 1 μολυσμένη. Η συνάρτηση εντοπίζει τη μολυσμένη ακμή uv και επιστρέφει τη σταθερά που χαρακτηρίζει τον κόμβο v (0 αν ο v είναι πατέρας του u, 1 αν ο v είναι αριστερό παιδί του u και 2 αν ο v είναι δεξί παιδί του u). Για να το πετύχει αυτό, αρχικοποιεί μία μεταβλητή i στο 0 (γραμμή 118) και επαναλαμβάνει την ακόλουθη διαδικασία: Διαβάζει την τιμή της i-οστής θέσης (θέση $i-1$) του πεδίου-πίνακα $isContaminated$ του κόμβου u ($getIsContaminated(u, i)$). Αν είναι 1, η μολυσμένη ακμή βρέθηκε και είναι αυτή που οδηγεί στον $(i+1)$-οστό γείτονα του κόμβου u. Τότε, η σύγκριση που λαμβάνει χώρα στη γραμμή 119 είναι ψευδής και η ροή του προγράμματος μεταφέρεται στη γραμμή 122, όπου η συνάρτηση επιστρέφει την τιμή της μεταβλητής i. Διαφορετικά, η σύκγριση είναι αληθής, οπότε η ροή μεταφέρεται στο σώμα του βρόχου (γραμμή 120), όπου η τιμή της μεταβλητής i αυξάνεται κατά 1 ($i++$).

- $slide(u, i)$ Η ολίσθηση ενός ερευνητή κατά μήκος μιας ακμής uv, όπου v ο i-οστός γείτονας του κόμβου u συνεπάγεται

  – την απομάκρυνση του ερευνητή από τον κόμβο u. Αυτό επιτυγχάνεται καταχωρίζοντας την τιμή 0 στο πεδίο $agent$ του κόμβου u ($setAgent(u, 0)$).

  – την άφιξή του ερευνητή στον κόμβο v, η διεύθυνση του οποίου βρίσκεται αποθηκευμένη στην i-οστή θέση (θέση $i-1$) του πεδίου-πίνακα $edge$ του κόμβου u (γραμμή 130, $v = getEdge(u, i)$). Αυτό επιτυγχάνεται καταχωρίζοντας την τιμή 0 στο πεδίο $agent$ του κόμβου v ($setAgent(v, 1)$).

- $isLeftChild(u, v)$ Δέχεται ως είσοδο τη διεύθυνση 2 κόμβων u, v και συγκρίνει τη διεύθυνση του αριστερού παιδιού του κόμβου u, την οποία διαβάζει από την 2η θέση (θέση 1) του πεδίου-πίνακα $edge$ του κόμβου u ($getEdge(u, 1)$), με την διεύθυνση του κόμβου v. Κατά συνέπεια, επιστρέφει 1 αν ο v είναι αριστερό παιδί του κόμβου u, αλλιώς επιστρέφει 0.

- $isRightChild(u, v)$ Δέχεται ως είσοδο τη διεύθυνση 2 κόμβων u, v και συγκρίνει τη διεύθυνση του δεξιού παιδιού του κόμβου u, την οποία διαβάζει από την 3η θέση (θέση 2) του πεδίου-πίνακα $edge$ του κόμβου u ($getEdge(u, 2)$), με την διεύθυνση του κόμβου v. Κατά συνέπεια, επιστρέφει 1 αν ο v είναι δεξί παιδί του κόμβου u, αλλιώς επιστρέφει 0.

Βιβλιογραφία

[1] Lali Barrière, Paola Flocchini, Fedor V Fomin, Pierre Fraigniaud, Nicolas Nisse, Nicola Santoro, and Dimitrios M Thilikos. Connected graph searching. *Information and Computation*, 219:1–16, 2012.

[2] Lali Barriere, Paola Flocchini, Pierre Fraigniaud, and Nicola Santoro. Capture of an intruder by mobile agents. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, pages 200–209. ACM, 2002.

[3] Lali Barriere, Pierre Fraigniaud, Nicola Santoro, and Dimitrios M Thilikos. Searching is not jumping. In *Graph-Theoretic Concepts in Computer Science*, pages 34–45. Springer, 2003.

[4] Daniel Bienstock and Paul Seymour. Monotonicity in graph searching. *Journal of Algorithms*, 12(2):239–245, 1991.

[5] Lélia Blin, Janna Burman, and Nicolas Nisse. Exclusive graph searching. In *Algorithms–ESA 2013*, pages 181–192. Springer, 2013.

[6] R Breisch. An intuitive approach to speleotopology. *Southwestern cavers*, 6(5):72–78, 1967.

[7] Paola Flocchini, Miao Jim Huang, and Flaminia L Luccio. Contiguous search in the hypercube for capturing an intruder. In *Parallel and Distributed Processing Symposium, 2005. Proceedings. 19th IEEE International*, pages 62–62. IEEE, 2005.

[8] Paola Flocchini, Miao Jun Huang, and Flaminia L Luccio. Decontaminating chordal rings and tori using mobile agents. *International Journal of Foundations of Computer Science*, 18(03):547–563, 2007.

[9] Paola Flocchini, Miao Jun Huang, and Flaminia L Luccio. Decontamination of hypercubes by mobile agents. *Networks*, 52(3):167–178, 2008.

[10] Fedor V Fomin, Dimitrios M Thilikos, and Ioan Todinca. Connected graph searching in outerplanar graphs. *Electronic Notes in Discrete Mathematics*, 22:213–216, 2005.

[11] Pierre Fraigniaud and Nicolas Nisse. Connected treewidth and connected graph searching. In *LATIN 2006: Theoretical Informatics*, pages 479–490. Springer, 2006.

[12] Jens Gusted. On the pathwidth of chordal graphs. *Discrete Applied Mathematics*, 45(3):233–248, 1993.

[13] Lefteris M Kirousis and Christos H Papadimitriou. Searching and pebbling. *Theoretical Computer Science*, 47:205–218, 1986.

[14] Andrea S LaPaugh. Recontamination does not help to search a graph. *Journal of the ACM (JACM)*, 40(2):224–245, 1993.

[15] Euripides Markou, Nicolas Nisse, and Stéphane Pérennes. Exclusive graph searching vs. pathwidth. 2014.

[16] Nimrod Megiddo, S Louis Hakimi, Michael R Garey, David S Johnson, and Christos H Papadimitriou. The complexity of searching a graph. *Journal of the ACM (JACM)*, 35(1):18–44, 1988.

[17] Rodica Mihai and Ioan Todinca. Pathwidth is np-hard for weighted trees. In *Frontiers in Algorithmics*, pages 181–195. Springer, 2009.

[18] Nicolas Nisse. Connected graph searching in chordal graphs. *Discrete Applied Mathematics*, 157(12):2603–2610, 2009.

[19] Torrence D Parsons. Pursuit-evasion in a graph. In *Theory and applications of graphs*, pages 426–441. Springer, 1978.

[20] Torrence D Parsons. The search number of a connected graph. In *Proc. 9th South-Eastern Conf. on Combinatorics, Graph Theory, and Computing*, pages 549–554, 1978.

[21] Neil Robertson and Paul D Seymour. Graph minors. i. excluding a forest. *Journal of Combinatorial Theory, Series B*, 35(1):39–61, 1983.

[22] Paul D Seymour and Robin Thomas. Graph searching and a min-max theorem for tree-width. *Journal of Combinatorial Theory, Series B*, 58(1):22–33, 1993.

[23] Boting Yang, Danny Dyer, and Brian Alspach. Sweeping graphs with large clique number. In *Algorithms and computation*, pages 908–920. Springer, 2005.

Παράρτημα Α΄

Υλοποίηση του αλγορίθμου μας σε C

Α΄.1 Ο κώδικας

Α΄.1.1 Το αρχείο BST.h

```
1  #ifndef BST_H
2  #define PARENT 0
3  #define LEFT_CHILD 1
4  #define RIGHT_CHILD 2
5
6  typedef struct node
7  {
8      int key;
9      int agent;
10     int nofContEdges;
11     int label;
12     int isContaminated[3];
13     struct node *edge[3];
14 }node;
15
16 typedef struct tree
17 {
18     node *root;
19 }tree;
20
21 void setRoot( tree *T, node *u );
22 void setKey( node *u, int val );
23 void setAgent( node *u, int val );
24 void setLabel( node *u, int val );
25 void setNofContEdges( node *u, int val );
26 void setEdge( node *u, int i, node *v );
27 void setIsContaminated( node *u, int i, int val );
28 node *getRoot( tree T );
29 int getData( node *u );
30 int getAgent( node *u );
```

```
31 int getLabel( node *u );
32 int getNofContEdges( node *u );
33 node *getEdge( node *u, int i );
34 int getIsContaminated( node *u, int i );
35 void init( tree *T );
36 void creation( tree *T );
37 int isEmpty( tree T );
38 node *newNode( int num );
39 node *insert( tree *T, int num );
40 node *insertAsLeaf( node *prev, node *cur, int x );
41
42 #endif
```

Α΄.1.2 Το αρχείο BST.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include "BST.h"
4
5 void setRoot( tree *T, node *u )
6 {
7     T->root = u;
8 }
9
10 void setKey( node *u, int val )
11 {
12      u->key = val;
13 }
14
15 void setAgent( node *u, int val )
16 {
17      u->agent = val;
18 }
19
20 void setLabel( node *u, int val )
21 {
22      u->label = val;
23 }
24
25 void setNofContEdges( node *u, int val )
26 {
27      u->nofContEdges = val;
28 }
29
30 void setEdge( node *u, int i, node *v )
31 {
32     u->edge[i] = v;
33 }
34
```

```
35 void setIsContaminated( node *u, int i, int val )
36 {
37     u->isContaminated[i] = val;
38 }
39
40 node *getRoot( tree T )
41 {
42     return T.root;
43 }
44
45 int getKey( node *u )
46 {
47     return u->key;
48 }
49
50 int getAgent( node *u )
51 {
52     return u->agent;
53 }
54
55 int getLabel( node *u )
56 {
57     return u->label;
58 }
59
60 int getNofContEdges( node *u )
61 {
62     return u->nofContEdges;
63 }
64
65 node *getEdge( node *u, int i )
66 {
67     return u->edge[i];
68 }
69
70 int getIsContaminated( node *u, int i )
71 {
72     return u->isContaminated[i];
73 }
74
75 void init( tree *T )
76 {
77     setRoot( T, NULL );
78     printf( "Tree initialization successfully completed\n" );
79 }
80
81 void creation( tree *T )
82 {
83     FILE *fp;
84     int x, count;
85
86     init( T );
```

```
87
88     fp = fopen( "input.txt","r" );
89
90     do
91     {
92         fscanf( fp, "%d", &x );
93         if( x != -1 )
94             insert( T, x );
95     }while( x != -1 );
96
97     printf( "Tree creation successfully completed\n" );
98 }
99
100 int isEmpty( tree T )
101 {
102     return getRoot( T ) == NULL;
103 }
104
105 node *newNode( int num )
106 {
107     node *new;
108     int i;
109
110     new = malloc( sizeof( node ) );
111
112     setKey( new, num );
113     setLabel( new, 0 );
114
115     for( i = 0; i <= 2; i++ )
116         setEdge( new, i, NULL );
117
118     for( i = 0; i <= 2; i++ )
119         setIsContaminated( new, i, -1 );
120
121     setNofContEdges( new, 0 );
122
123     return new;
124 }
125
126 node *insert( tree *T, int num )
127 {
128     node *previous, *current, *ptr;
129
130     ptr = newNode( num );
131
132     if( isEmpty( *T ) )
133         setRoot( T, ptr );
134     else
135     {
136         current = getRoot( *T );
137         while( current != NULL )
138         {
```

```
139             previous = current;
140             if( num < getKey( current ) )
141                 current = getEdge( current, LEFT_CHILD );
142             else
143                 current = getEdge( current, RIGHT_CHILD );
144         }
145 
146         if( getKey( ptr ) < getKey( previous ) )
147             ptr = insertAsLeaf( previous, ptr, LEFT_CHILD );
148         else
149             ptr = insertAsLeaf( previous, ptr, RIGHT_CHILD );
150     }
151 
152     printf("Insertion successfully completed\n");
153 
154     return ptr;
155 }
156 
157 node *insertAsLeaf( node *prev, node *neos, int i )
158 {
159     int contEdges;
160     setEdge( prev, i, neos );
161     setIsContaminated( prev, i, 1 );
162     contEdges = getNofContEdges( prev );
163     contEdges++;
164     setNofContEdges( prev, contEdges );
165     setEdge( neos, PARENT, prev );
166     setIsContaminated( neos, PARENT, 1 );
167     contEdges = getNofContEdges( neos );
168     contEdges++;
169     setNofContEdges( neos, contEdges );
170 
171     return neos;
172 }
```

Α΄.1.3 Το αρχείο placement.h

```
1 #ifndef placement_H
2 
3 int postorderLabeling( node *u, int nofLnodes );
4 int putSearchers( node *u, int isLeft, int nofSearchers );
5 int nofLabeledKids( node *u );
6 int nofUnlabeledKids( node *u );
7 int isLabeled( node *u );
8 
9 #endif
```

Α΄.1.4 Το αρχείο placement.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "BST.h"
4  #include "placement.h"
5
6  int postorderLabeling( node *u, int nofLnodes )
7  {
8      if( u != NULL )
9      {
10         nofLnodes = postorderLabeling( getEdge( u, LEFT_CHILD ), ↩
               nofLnodes );
11         nofLnodes = postorderLabeling( getEdge( u, RIGHT_CHILD ), ↩
               nofLnodes );
12
13         if( nofUnlabeledKids( u ) == 2 || getEdge( u, PARENT ) == NULL↩
               )
14         {
15             setLabel( u, 1 );
16             nofLnodes = nofLnodes + 1;
17         }
18     }
19     return nofLnodes;
20 }
21
22 int putSearchers( node *u, int isLeft, int nofSearchers )
23 {
24     if( u != NULL )
25     {
26         if( isLabeled( u ) )
27         {
28             if( getEdge( u, PARENT ) == NULL && ( getEdge( u, ↩
                   LEFT_CHILD ) == NULL || getLabel( getEdge( u, ↩
                   LEFT_CHILD ) ) == 1 ) )
29             {
30                 setAgent( u, 1 );
31                 nofSearchers = nofSearchers + 1;
32             }
33             nofSearchers = putSearchers( getEdge( u, LEFT_CHILD ), 1, ↩
                   nofSearchers );
34             nofSearchers = putSearchers( getEdge( u, RIGHT_CHILD ), 0,↩
                   nofSearchers );
35         }
36         else
37         {
38             if( nofUnlabeledKids( u ) == 0 && isLeft == 1 )
39             {
40                 setAgent( u, 1 );
41                 nofSearchers = nofSearchers + 1;
```

```
42              }
43              nofSearchers = putSearchers( getEdge( u, LEFT_CHILD ), ↩
                    isLeft, nofSearchers );
44              nofSearchers = putSearchers( getEdge( u, RIGHT_CHILD ), ↩
                    isLeft, nofSearchers );
45          }
46      }
47      return nofSearchers;
48  }
49
50  int nofLabeledKids( node *u )
51  {
52      int labeledKids = 0;
53      node *v;
54
55      v = getEdge( u, LEFT_CHILD );
56      if( v != NULL )
57          if( getLabel( v ) == 1 )
58              labeledKids = labeledKids + 1;
59
60      v = getEdge( u, LEFT_CHILD );
61      if( v != NULL )
62          if( getLabel( v ) == 1 )
63              labeledKids = labeledKids + 1;
64
65      return labeledKids;
66  }
67
68  int nofUnlabeledKids( node *u )
69  {
70      int unlabeledKids = 0;
71      node *v;
72
73      v = getEdge( u, LEFT_CHILD );
74      if( v != NULL )
75          if( getLabel( v ) == 0 )
76              unlabeledKids = unlabeledKids + 1;
77
78      v = getEdge( u, RIGHT_CHILD ) ;
79      if( v != NULL )
80          if( getLabel( v ) == 0 )
81              unlabeledKids = unlabeledKids + 1;
82
83      return unlabeledKids;
84  }
85
86  int isLabeled( node *u )
87  {
88      return u == NULL || ( u != NULL && getLabel( u ) == 1 );
89  }
```

Α΄.1.5 Το αρχείο cleaning.h

```
1  #ifndef cleaning_H
2
3  void postorderCleaning( node *temp );
4  void preorderCleaning_BT( node *temp );
5  void afterSlide( node *u, int i, node *v, int j );
6  int findTheContaminatedEdge( node *u );
7  int isLeftChild( node *u, node *v );
8  int isRightChild( node *u, node *v );
9  void cleaningEdge( node *u, int i );
10 int beforeSlide( node *u );
11 node *slide( node *u, int i );
12 void searchingOfNeighboringNodes( node *u );
13
14 #endif
```

Α΄.1.6 Το αρχείο cleaning.c

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "BST.h"
4  #include "cleaning.h"
5
6  void postorderCleaning( node *u )
7  {
8      int contEdge;
9      node *v;
10
11     if( u != NULL )
12     {
13         postorderCleaning( getEdge( u, LEFT_CHILD ) );
14         postorderCleaning( getEdge( u, RIGHT_CHILD ) );
15
16         if( getLabel( u ) == 0 && getAgent( u ) == 1 )
17         {
18             contEdge = beforeSlide( u );
19             v = slide( u, contEdge );
20             if( isLeftChild( v, u ) )
21                 afterSlide( u, contEdge, v, LEFT_CHILD );
22            else if( isRightChild( v, u ) )
23                afterSlide( u, contEdge, v, RIGHT_CHILD );
24             else
25                 afterSlide( u, 0, v, PARENT );
26             searchingOfNeighboringNodes( v );
27         }
28     }
```

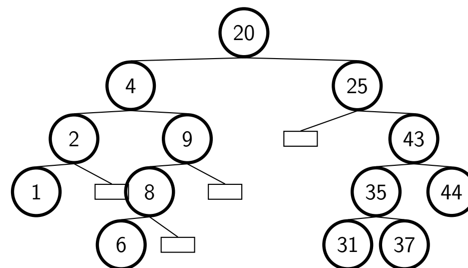
```
29 }
30
31 void preorderCleaning( node *u )
32 {
33     int contEdge;
34     node *v;
35
36     if( u != NULL )
37     {
38         if( getNofContEdges( u ) == 1 )
39         {
40             contEdge = beforeSlide( u );
41             v = slide( u, contEdge );
42             if( isLeftChild( v, u ) )
43                 afterSlide( u, contEdge, v, LEFT_CHILD );
44             else if( isRightChild( v, u ) )
45                 afterSlide( u, contEdge, v, RIGHT_CHILD );
46             else
47                 afterSlide( u, contEdge, v, PARENT );
48             searchingOfNeighboringNodes( v );
49         }
50
51         preorderCleaning( getEdge( u, LEFT_CHILD ) );
52         preorderCleaning( getEdge( u, RIGHT_CHILD ) );
53     }
54 }
55
56 void afterSlide( node *u, int i, node *v, int j )
57 {
58     FILE *fp;
59
60     cleaningEdge( u, i );
61
62     fp = fopen( "results.txt", "a" );
63     fprintf( fp, "%d -> %d\n", getKey( u ), getKey( getEdge( u, i ) ) ↩
           );
64     fclose( fp );
65
66     cleaningEdge( v, j );
67 }
68
69 void searchingOfNeighboringNodes( node *u )
70 {
71     int k;
72     FILE *fp;
73
74     for( k = 0; k <= 2; k++ )
75         if( getEdge( u, k ) != NULL && getAgent( getEdge( u, k ) ) == ↩
               1 )
76         {
77             cleaningEdge( u, k );
78
```

```
79              fp = fopen( "results.txt", "a" );
80              fprintf( fp, "%d <-> %d\n", getKey( u ), getKey( getEdge( ↩
                   u, k ) ) );
81              fclose( fp );
82
83              if( k == PARENT )
84              {
85                  if( isLeftChild( getEdge( u, PARENT ), u ) )
86                      cleaningEdge( getEdge( u, PARENT ), LEFT_CHILD );
87                  else
88                      cleaningEdge( getEdge( u, PARENT ), RIGHT_CHILD );
89              }
90              else
91                  cleaningEdge( getEdge( u, k ), PARENT );
92          }
93  }
94
95  void cleaningEdge( node *u, int i )
96  {
97      int contEdges;
98
99      setIsContaminated( u, i, 0 );
100     contEdges = getNofContEdges( u );
101     contEdges--;
102     setNofContEdges( u, contEdges );
103 }
104
105 int beforeSlide( node *u )
106 {
107     int i;
108
109     i = findTheContaminatedEdge( u );
110
111     return i;
112 }
113
114 int findTheContaminatedEdge( node *u )
115 {
116     int i;
117
118     i = 0;
119     while( getIsContaminated( u, i ) != 1 )
120         i++;
121
122     return i;
123 }
124
125 node *slide( node *u, int i )
126 {
127     node *v;
128
129     setAgent( u, 0 );
```

```
130     v = getEdge( u, i );
131     setAgent( v, 1 );
132 
133     return v;
134 }
135 
136 int isLeftChild( node *u, node *v )
137 {
138     return getEdge( u, LEFT_CHILD ) == v;
139 }
140 
141 int isRightChild( node *u, node *v )
142 {
143     return getEdge( u, RIGHT_CHILD ) == v;
144 }
```

Α΄.2 Παράδειγμα εκτέλεσης κώδικα

Σε ένα αρχείο $input.txt$ αποθηκεύουμε ακεραίους, τον έναν δίπλα στον άλλον. Μεταξύ 2 ακεραίων αφήνουμε ένα κενό. Ο κάθε ακέραιος αντιπροσωπεύει το κλειδί ενός στοιχείου που θα αποθηκευτεί στο δυαδικό δένδρο. Έστω, για παράδειγμα ότι θέλουμε να δώσουμε την εξής είσοδο στο πρόγραμμά μας: Γιατί διαλέξαμε το συγκεκριμένο δένδτρο; Τυχαία.



Θα μπορούσε να είναι οποιοδήποτε άλλο. Για την ακρίβεια η επιλογή μας δεν προέκυψε ακριβώς στην τύχη. Είναι το δένδρο που χρησιμοποιήσαμε στα κεφάλαια $3-6$, οπότε ξέρουμε ακριβώς τη διαδικασία με την οποία καθαρίζουν οι ακμές του. Έτσι, το αρχείο $input.txt$ θα έχει τη μορφή:

20 4 25 2 9 43 1 8 35 44 6 31 37

Τι θα πάρουμε ως έξοδο; Ένα αρχείο $results.txt$ που θα μας δίνει πληροφορία για την σειρά αλλά και τον τρόπο με τον οποίο καθάρισαν οι ακμές του δέντρου. Στο συγκεκριμένο παράδειγμα, το αρχείο $results.txt$ θα έχει την εξής μορφή:

$$1 \to 2$$
$$2 \to 4$$
$$4 \leftrightarrow 20$$
$$31 \to 35$$

$$20 \to 25$$
$$4 \to 9$$
$$9 \to 8$$
$$8 \to 6$$
$$25 \to 43$$
$$43 \leftrightarrow 35$$
$$43 \to 44$$
$$35 \to 37$$

όπου:

- $a \to b$ σημαίνει ότι η ακμή ab καθάρισε ύστερα από ολίσθηση ενός ερευνητή κατά μήκος αυτής, ενώ
- $a \leftrightarrow b$ σημαίνει ότι η ακμή καθάρισε αφού υπήρχε ερευνητής και στα δυο της άκρα ταυτόχρονα.